RADC-TR-81-137
Final Technical Report
June 1981

LEVEL II

12


AD A108645

# DATA INDEPENDENT ACCESSING METHODOLOGY DISTRIBUTED ACCESS

Sterling Systems, Inc.

Lowell I. Schneider
Michael Levin

12 242

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED



ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441



81 12 14 132

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-137 has been reviewed and is approved for publication.

APPROVED: Patricia M Langendorf

PATRICIA M. LANGENDORF
Project Engineer

APPROVED:

JOHN N. ENTZMINGER
Technical Director
Intelligence & Reconnaissnce Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRDA) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|
| 1. REPORT NUMBER: RADC-TR-81-137 | 2. GOVT ACCESSION NO.: AD-A108645 |
| 3. RECIPIENT'S CATALOG NUMBER | |
| 4. TITLE (and Subtitle): DATA INDEPENDENT ACCESSING METHODOLOGY DISTRIBUTED ACCESS | 5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report |
| | 6. PERFORMING ORG. REPORT NUMBER: N/A |
| 7. AUTHOR(s): Lowell I. Schneider, Michael Levin | 8. CONTRACT OR GRANT NUMBER(s): F30602-79-F-0054 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: Sterling Systems, Inc., 14618 West 6th St., Golden CO 80401 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 45941628 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (IRDA), Griffiss AFB NY 13441 | 12. REPORT DATE: June 1981 |
| | 13. NUMBER OF PAGES: 283 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same | 15. SECURITY CLASS. (of this report): UNCLASSIFIED |
| | 15a. DECLASSIFICATION DOWNGRADING SCHEDULE: N/A |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Patricia M. Langendorf (IRDA)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data Base Management
Distributed Processing
DIAM
Data Independent Accessing Model
Enterprise Modeling
Algebraic Graph Theory

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DIAM, The Data Independent Accessing Model, is a theory for describing the various ways in which information can be represented in a computer. This report uses a relational model of data to define the information resources of a collection of data archives, and the details of how it is stored to the extent necessary to extract it. Access begins by asking a question of the Relational Model (which cuts across sources in

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

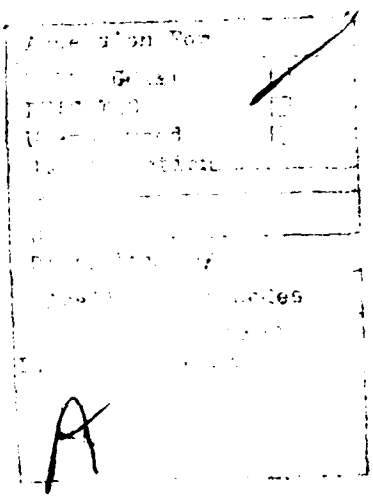
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

4 0 1 1 1

combination).

The DIAM model then makes it possible to decompose the question into parts so small that no more than one source is needed to answer each part, translate each part to correspond to the representation used by the source, and recompose the translated parts into the largest possible pieces, each of which corresponds to a common source. These requests are then transmitted to the required sources and the returned information is decomposed and recomposed in precisely the opposite way that the questions was, yielding the originally sought answer.

The report concludes that it is now both feasible and justified to develop a prototype data base access compiler for experimental testing in existing networks.



TECHNICAL REPORT SUMMARY

## 1.0 INTRODUCTION

Intelligence has frequently been described as "the attempt to reconstruct a real-world scenario based on imperfect information from unreliable sources." In perspective the same can be said of any information gathering enterprise, be it the military or a shoe store. The moment we replace real-world objects with any kind of surrogate such as a plastic ship on a chart table or a sequence of symbols on paper, the real-world becomes an absolute truth which we can only approximate from then on. There is no vocabulary or schematic rich enough to capture all the complexity and subtleness that the real world contains; and no method powerful enough to capture and record all the things that can happen in it.

The advent of automated data processing was, in retrospect, both the greatest blessing and the worst evil to ever befall the information management problem. The speed and capacity with which even the early computers could acquire, process and store data, enamoured everyone and this quality has grown to bewildering proportions today. The positive outgrowths of this technology are desiderata and suffice it to say that our society could not function without them. But there was a hidden cost, particularly to the information management community, that was slow to be revealed and its ultimate magnitude is yet to be appreciated. The sacrifice of fidelity that accompanied the use of surrogates like words, pictures or plastic models was magnified a thousand fold by the

utterly rigid and absolute way in which computers did and still do represent information. It is very difficult for a person to detect any difference between two cards with the word "computer" starting in column 35 on one and column 36 on the other. It is very difficult for a machine to detect any similarity.

Several decades of computer language development preceeded any noticable advances translating from the relative way people perceive things into the absolute mechanisms by which machines recognize them. The fairly recent combination of word processing and graphics software is the first time the functionality of a mere pencil and eraser could be emulated. And the evolution of software we now call database systems has finally allowed people to record and inquire about information in relation to something else. While both of these capabilities remain primitive, and are realized by a collection of ingenious gimmicks rather than theory, they share these important facts:

A. They present information according to a relative model with which a person is familiar;

B. They represent the information to a machine in the absolute form it can process most efficiently;

C. Somehow they can get from one to the other.

And recognition of this has definitely caught theoretic attention.

In the first place, the person's relational model and the machine's representation tend to be quite different, and change at

different times and for different reasons. To the extent that one could be insulated from the other by generalizing the means for translating between them, more efficient machine representations can be sought without impacting people who use the information; and investigatiors studying the fidelity of information with respect to the real world can do so unhampered by the details of computer representations. In the second place, the mere fact of being able to transform information from one representation to another without changing its meaning implies that representations in general must obey some underlying conservative principles. If these could be discovered, precise definition of the gimmickry of translation and the inevitable inconsistency or incompleteness of one representation with respect to another would both be possible.

This theoretic work, in turn, has gained pragmatic attention. Information is most frequently collected by source but accessed by subject. Collecting agencies have relied heavily on things like database systems to improve the efficiency and stability of their operations. But by the inventional nature of the relative models these systems present, diversity among sources is an increasingly difficult obstacle to multi-source access. Concurrently, the relational nature of these models increasingly reveals and motivates additional multi-source correlation. While it is always possible to construct a correlation for a specific combination of

sources by sheer ingenuity, the continual proliferation of new sources and the desire to access them in combination will, at some point, have to be resolved by an approach which is comprehensive and complete with respect to representations in general.

One such approach has been under investigation by RADC. It is formed on the coupling of two rather important theories developed in the database research community:

A) The Relational Model of Data [3] is a theory for describing the information contained in a data bank without depending on any knowledge of how it is represented in the computer.

B) DIAM, The Data Independent Accessing Model [2], is a theory for describing the various ways in which information can be represented in a computer.

The coupling is achieved by injecting the mathematical operators of the Relational Model of Data (known as relational algebra) into the DIAM structure. One important application of this coupling sought by RADC is the delegated production system depicted in Figure 1.

Given a collection of information sources, the Relational Model is used to define the information resource of the whole collection. DIAM is used, in turn, to define what part of it each source contains, and the details of how it is stored to the extent necessary to extract it. Access begins by asking a question of the Relational Model (which cuts across sources in combination). By virtue of their mathematical coupling, it is then possible to:

1) Decompose the question into parts so small that no more than one source is needed to answer each part (several sources might qualify singularly).

2) Translate each part to correspond to the representation used by the source.

3) Recompose the translated parts into the largest possible pieces, each of which corresponds to a common source.

These requests are then transmitted to the required sources and the returning information is decomposed and recomposed in precisely the opposite way that the question was, yielding the orginally sought answer.

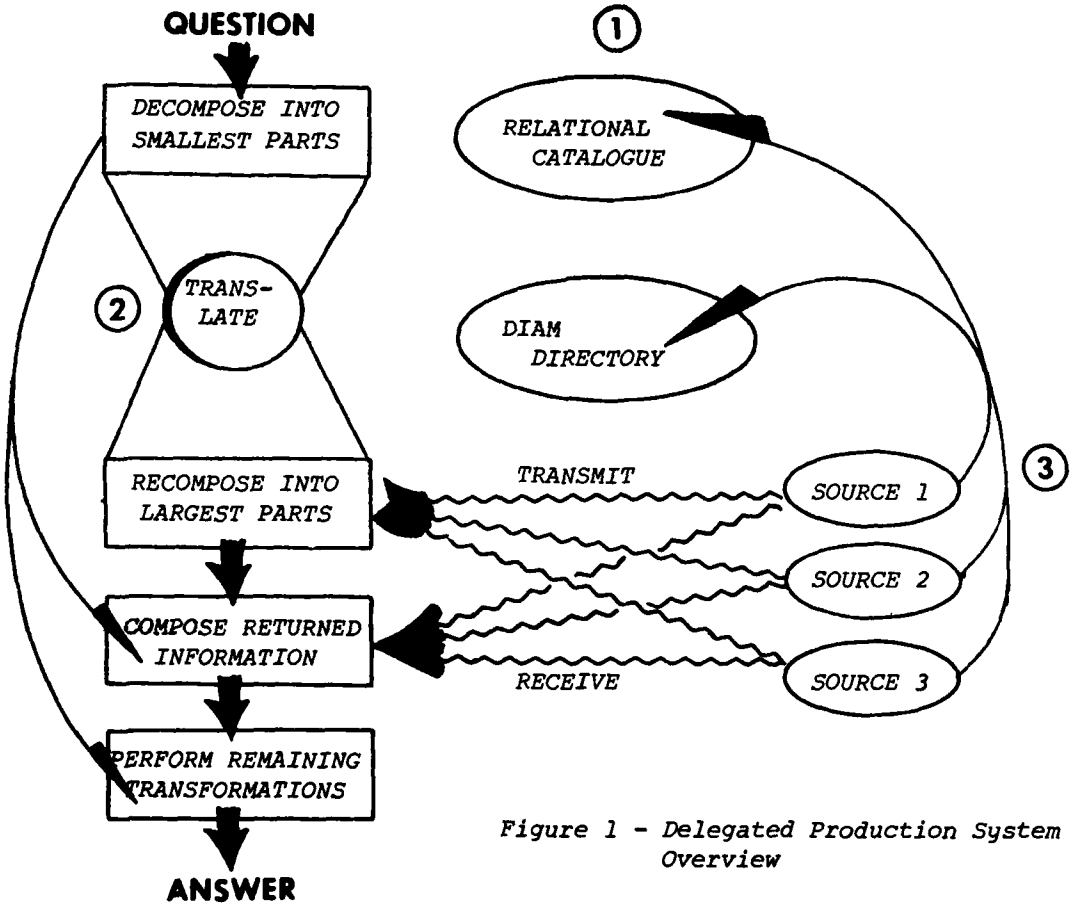



*Figure 1 - Delegated Production System Overview*

The development of the delegated production technology has been slow and tedious, as will be discussed in 2.0 - BACKGROUND. But there has been a growing conviction that it is sufficiently mature to be prototyped. This report conveys the results of an investigation to identify and, if possible, resolve any issues that might prevent a prototype effort from being properly justified at this stage of developmennt. A number of suspected problems were enumerated at the outset of the investigation, but they tended to cogulate into two major theoretic issues and one pragmatic issue - neither of which could have been predicted. Referring again to Figure 1, these issues are:

1. Constructing a relational model that behaves as a universal umbrella to interrelate multi-source data is theoretically possible due to a mathematical formalism called the UR (Universal Relation) assumption. In practice, however, real world amorphousness is far richer and complex than this assumption can support and Volume I proposes that it be replaced by a semantic reference model that deals with all the real world entities rather than just the symbolic subset that are storeble. (It is interesting to note that in [7], it was recently shown that the UR-assumption failed in even the simplest cases as well).

2. The DIAM structure into which the discipline of known relational mathematics had been injected was, itself, not a known mathematical theory. It very closely resembled graph theory (the theory of networks consisting of nodes and paths connecting them) and its purpose - the navigation of paths among sources - was identical. But it employed typed edges; i.e., not all connections obeyed the same rules as graph theory requires. It was therefore designated string theory and represented a "new math" to which the known results of graph theory could not be applied. This necessitated the use of algoritums - rather than theorems - which are difficult to implement efficiently and even more difficult to describe. Volume II demonstrates an elimination of the typed edge concept that concurrently increases the power of the theory.

3. The collection of information into the resource directory that drives the delegated production system is a highly technical discipline. This is not problematic in a laboratory environment where theorists who understand the technology perform the task. In reality, logistics clearly require that analysts quite unfamiliar with the theory will have to do this themselves, and without even the help of consultants if sensitive information is involved. Volume III is an attempt to produce a "cookbook" approach

to the problem. Unfortunately, it was done before the ideas in Volume I had been formulated and will need updating once more.

## 2.0 BACKGROUND

The late Dr. Michael E. Senko was probably the first to actively pursue the belief that the representation of information in physical space did, in fact, obey an underlying theory. After an unknown number of years of development under the auspices of IBM Research, RADC contracted with IBM to apply this concept to improving the design of data handling systems. The principal result of this contract, published in the late sixties, was the File Organization and Evaluation Model (FOREM). Motivated by the ideas implemented a decade earlier in IBM's Formatted File System, i.e., files whose content and organization were wholly contained within them, FOREM was the first demonstration that information could be systematically separated from its representation. FOREM was rather narrow in scope, dealing primarily with a generalization of IBM-like access methods. But it must be remembered that database systems, as we know them today, were not yet in very widespread use.

FOREM was never applied to any significant extent because of these limitations and the results of a few empirical experiments reported in Communications of the Association for Computing machinery (CACM) are the only traces it left. But the fact of the separation - that representations could be treated as an independent variable - was a profound motivation for IBM. A team of prominent, scientists was assembled under Dr. Senko's direction to develop what was then called the Universal Information Systems Technology

(UIST). The project was classified by IBM as proprietary making it difficult to trace its activities or membership. But it is generally believed that Dr. E.F. (Ted) Codd, who later rose to unprecedented prominence from his now famous publication on the Relational Model [9], was an important contributor.

For unknown reasons, UIST was declassified in December 1972 and this was accompanied by the release of IBM Research Report RJ982[2]. The theory reported was renamed "A Data Independent Accessing Model" and was followed a year later by a three part article in IBM Systems Journal. Sometime during this period the project was also disbanded. The journal article was not well received due, primarily, to the ulra-logical but incomprehensible terminology and only a few researchers willing to master it found that, underlying the journal report, was a virtual treatise of nearly a thousand pages detailing the discoveries of UIST. Not to discredit the efforts of these researchers, there was only one that is directly pertinent to this report.

In 1973, the National Aeronautics and Space Administration awarded a contract to Martin Marietta Corporation to develop a generalized database system simulator. Martin Marietta was the only contractor to propose DIAM as the foundation for the simulator. It was also the first known attempt to deploy DIAM theory in application. In close collaboration with Dr. Senko, Martin Marietta spent two years in developing a direct implementation of the theory

as described in RJ982. As such, the simulator was not analytic, i.e., formulae constructed by curve fitting empirical performance data. It was a discrete-event model which emulated the detailed actions of a database system that DIAM theory predicted would have to occur. At the RJ982 stage of development, the theory was rough, even by Dr. Senko's admission, and the experimental testing revealed this. Against the real system modelled, a controlled experiment produced a fidelity of $\pm .5$ which was insufficient for NASA's use but on the other hand for exceeded Dr. Senko's estimate of $\pm$ 3.

NASA dropped the project in 1975 but a number of factors, not the least of which was Dr. Senko's persuasion, motivated Martin Marietta to continue the research under its own Independent Research and Development Program, which lasted until 1977. The major failing of DIAM theory addressed was that it was too permissive: it allowed several possible explanations of the same phenomenon without any rationale for which was correct. The major advance in this effort was the discovery that the somewhat generic typing of access paths in RJ982 could be replaced with a highly restrictive typing based on Codd's relational algebra - without loss of generality. The result was that there was one and only one description for any access structure; and the simulator was redesigned completely around this concept. A subsequent retest against the same, and one other system, demonstrated sufficient fidelity that the vendors of the system requested that the results not be published for proprietary reasons.

Dr. Senko was somewhat chagrined that the relational model, with which he was not in agreement, had so vastly improved the DIAM theory. But his continued encouragement turned the project toward the more lofty goal of not just predicting, but actually directing the behaviour of database systems. To attract needed support for further advancement of the theory that was quite properly his discovery, he recalled on RADC who had begun sponsorship of the work nearly a decade earlier. This lead to a contract between RADC and Martin Marietta to verify that the progress Dr. Senko had reported was, indeed, a reality. The objective was to emulate the behaviour, in extensive detail, of a complicated Air Force system that exhibited as many non-standard features as could be found. The hypothesis, of course, was that if the theory could describe all these anomalies in concert, it could probably describe any database system, commercial or otherwise. The results are reported in "DIAM PACER Analysis" [5] and were sufficiently convincing. But the sheer size and complexity of the model - which had now grown to 60,000 lines of FORTRAN, necessitated a feasibility study before any decision to prototype the system could be considered.

In 1978, an RADC procurement was initiated with Sterling Systems, Incorporated, where the principal investigator of the Martin project was then employed, to perform such a study. A contract was consumated in mid 1979 and concluded in October 1980 and this report is a summary of the findings. The outset objectives were an enumeration of the areas of concern:

A) The power of the language the system could compile was known to be limited with respect to quantification and difference and a number of proposed extensions were to be examined.

B) The computational resources needed to operate the system were undefined, but were empirically well beyond machines for which RADC would like the system targeted. The goal was to determine the resources needed and, if possible, suggest techniques for reducing them.

Another concern, the ability of the Relational Model to capture the semantic richness and complexity of proposed application areas, was being dealt with intensively by Dr. Senko in his development of the entity - based FORAL model and language (FORAL is not an acronym), and was not included initially. His untimely death in December 1978 clearly incorporated this issue as another objective under the heading of "knowledge management".

As discussed in the introduction, each of these concerns were addressed and somewhat resolved, but in a combination that could not have been predicted in advance.

## 3.0 THE SEPARATION OF ENTERPRISE AND DATA MODELS (VOLUME I) SUMMARY

The proposed delegated production system rests on the assumption that a common relational umbrella interrelates the information among the various participating sources. While there is no problem in principle with creating a relational model of a database whose schema is given explicitly, a "good" relational model; i.e., one that obeys prescribed normal forms, requires knowledge that the schema alone does not contain. This problem is intensified when one attempts to integrate a multitude of existing sources (as opposed to designing a new network of sources) because of the differences in perception and convention from source to source. The principal problems cluster in two areas: naming and attributes.

Names have rarely been considered a problem in database design. A designer usually has the option to impose names or identifiers at will, eventhough they may at times be artifical (e.g., an employee number). But when one attempts to integrate existing databases, he must passively accept the names each of the designers has imposed, even though they may be different and inconsistent. One source may identify persons by social security number; another by serial number; and yet a third by first, middle, last name and date of birth. In a multi-source environment, names will be partial or overlapping and any assumption to the contrary is unrealistic. Yet the relational model is a model of only names, and it therefore lacks the power to overcome these differences. If one source uses

a social security number and another a serial number, the Relational Model will always imply the existence of two people - even though the numbers may name the same person.

The recorded attributes (descriptors) of a real-world entity may vary from source to source. A command may record only the "names", rank and years of service of its personnel whereas a unit may record a detailed skills inventory and service record. Similarly, one wing might keep a detailed log of each plane's itineraries; another merely the total number of hours since major overhaul. This problem generalizes (and has even been called the generalization problem) when, for example, the service histories for transports are less detailed than those for strategic bombers, even though both are planes (unless transports may also include trucks!) But in its limitation to modelling only the storable symbolic subset of real world entities, the Relational Model is again unable to deal with this kind of amorphousness. The symbolic attributes in a Relational Model of planes must be homogeneous from plane to plane - even though such is not the case in reality. The idea of a universal relation - one with every possible attribute and "null" values for the attributes not applicable - fails quickly. When little enough is known so that the recorded information about two planes is identical, the Relational Model will recognize only a single plane when, in fact, there are two.

The problem distills to a simple realization: the correspondence of relationships among real entities to relations among the symbolic subset we use to name them is not always precise. Furthermore, it is not necessarily the fault of the designer in his choice of naming or descriptive schemes. The complexity of the real entities cannot in general be captured by the highly limited symbolic subset. Yet it has been properly recognized that this subset is the only one we can store and manipulate.

This volume of the report does not purport to solve the general semantic problem that "the song, the name of the song, and what the name of the song is called" are and will always be different. But it proposes that this inherent restriction on modelling information is not as severe as one might think. Since language possesses generic as well as individual names, there is no theoretic problem with making a model of the real-world entities in general, rather than the symbolic subset to which we have traditionally restricted ourselves. Admittedly, such a model will never be directly translatable to the storage media with which we are accustomed. But developing one serves two important purposes for integrating a multi-source environment.

First, it diagrams the scheme(s) by which names have been assigned to objects. This might sound trivial but without such a model, these schemes have been extremely covert; known only to the designer because he had no language in which to document them. Knowing them explicitly permits the integrator to decide what names

are available, and which are appropriate to use under given circumstances.

Second, it enables one to make precise assertions about what a particular name relation does (or does not) contribute to the enterprise being modelled. As such, it addresses the issues of completeness and consistency of stored information with respect to the enterprise (rather than with respect to itself as many relational modellers have felt is sufficient).

The modelling technology presented is not at all new. The focus is not on how to model, but rather on what to model regardless of technique. As such, it is an operational presentation that shows how existing modelling technology can be applied to overcoming the problems of name and attribute by merely referring to the underlying entities.

The methodology presented is not at all algorithmic (although this property would be desirable at a future time). It shows, mostly by example, that a separate model of the enterprise is an essential reference point for integrating information sources into a coherent relational model.

In summary, the results are incomplete in an idealistic sense, but tremendously useful in their present state. Separating the enterprise and data models does not make multi-source integration easy - it merely makes it possible.

# 4.0 THE ALGEBRAIC ACCESS GRAPH (VOLUME II) SUMMARY

The point at which DIAM couples to the Relational Model is called the "String Model." It was mentioned in the introduction that a string model differs from a conventional graph in that the edges or paths that connect the nodes are typed; i.e., they do not all obey the same rules. In the RJ982 version of DIAM, the types were such that it was not possible to compute mathematically the information that was traversed by following a path of strings. Relational DIAM [1] solved this problem by typing the strings with operators of the relational algebra, such that when a path was traversed, the operators were combined in exactly the same way to yield an algebraic expression of the information discovered. This made possile, but not very practical, the delegated production system that was demonstrated.

The first source of impracticality was that, in general, paths in existing database implementations are complex; they do not generally represent a single algebraic operation because their designers never knew about the relational algebra (it hadn't been invented). Consequently, a path that was natural to a designer required tedious decoding into its primitive algebraic operators in order to yield a DIAM model.

The second source was not the algebra of the relations, but that of the strings. The purpose of the string model was to enumerate the traversals that would discover the information from

which the answer to an inquiry could be extracted; and to do so in an "optimal" way. Even the simplest tools of graph theory such as a transition matrix (which when raised to the nth power computes all points that can be reached in n steps) were inaccessible because of the typed edge concept.

The algebraic theory in Volume II presents a multi-faceted breakthrough founded on a simple but elusive realization. If a string can be typed by an algebraic operator, it can be typed by an expression of operators. This being the case, an expression is like any other expression and there is only one type. Nothing is lost; expressions of operators combine in the same way that the operators do. But an enormous amount is gained.

The edges are homogenized to produce a labelled digraph (a directional graph with edges labelled by algebraic experssions). Thus the transition matrix can simultaneously compute the connectivity, the compounded expressions, and the cardinality of all traversals pertinent to an inquiry.

The necessity to decode implementations into unnatural primitives is eliminated. Any path computes a unique set of information and the computation can always be described by an algebraic expression.

In addition, several unanticipated results are achieved as well:

A) The question of language power with respect to quantification and difference is bounded by known theory which defines the class of expressions that are computable.

B) The thoery is closed at a point that makes its equally applicable to non-Von Neumann architectures such as data-base machines and associative memories. This was not true of string theory since it necessarily considered the order of operations.

C) The decomposition, which previously descended to the primitive operators, can now stop at the highest level equivalent expression.

D) While it is yet to be demonstrated, it is intuitively apparent that any algebra can be accomodated making the theory equally suitable for other than the Relational Model.

In summary, the DIAM theory of Algebraic Access Graphs has advanced the realization of the delegated production system far beyond what was originally expected.

## 5.0 DATABASE INVENTORY (VOLUME III) SUMMARY

To logistically deploy a delegated production system, particularly in sensitive environments, an enormous amount of descriptive information must be gathered to form the resource directory by which the system operates. It has already been stated that an information model independent of representation detail must be developed to describe the contents of each source. Implicit in this statement is that quantitative as well as qualitative descriptions are needed if the system is to resolve multi-source inquires efficiently. Since both descriptions must be at the representation-independent level; and personnel familiar with these aspects are not likely to be computer scientists; it is extremely important that a tutorial requiring the minimal technical knowledge be made available.

Volume III of this report is an attempt at a cookbook appraoch to inventorying a data source to the extent necessary to participate in a delegated production system. It is subdivided into two major sections on qualitative and quantitative aspects respectively. As might be expected, the qualitative section deals with the problem of creating a good relational model from an existing database. It does not benefit from the enterprise model concept reported in Volume I due simply to chronology. On the other hand, to minimize technical issues, it approaches normalization from a perspective of common sense rather than mathematics. Consequently the reader is

likely to reap much of the benefit if Volume I has been read; and a direct coupling to the enterprise model concept would be a rather easy revision which can and should take place.

The quantitative section deals with the static and dynamic statistics of a source with only a minimal reliance on statistical concepts. In addition, it is structured for a graceful progression in understanding in that the results of a first iteration will not be obseleted by more thorough analysis. Each change contributes to instead of negating earlier observations.

In summary, it is worth noting that while Volume III was developed as an integral part of the delegated production concept, it also stands alone as a philosophy and technique for analyzing or designing a database. In this regard it is somewhat remarkable in that it is the only approach that allows one to define a database problem; without resorting to stating the problem in terms of its solution.

## 6.0 CONCLUSIONS AND RECOMMENDATIONS

Because of the algebraic access path model, it is now possible to build a prototype distributed heterogeneous database query compiler that will operate on a reasonably small machine with good efficiency. Recall that this software will accept a request for information which is to be found in several distinct and dissimilar databases, and will plan and execute a method for retrieving this information and presenting it to the requestor for further data processing.

The development of this technology into application should be of great interest as a means of interfacing existing applications systems such as IDHS-80 while maintaining stability of these applications even while reorganizing or redesigning.

The suggested prototype distributed heterogeneous database query compiler would have the following features:

1. It will accept queries in a representative-independent language based on relational algebra. The first prototype compiler will process the algebraic operators select, project, join and union. A later version will also process the operations difference, count and sum.

2. It will make use of the algebraic theory of access paths. The algorithm for compilation will be expressed in this theory,

making it easier to describe, much more efficient to execute, and resulting in a more portable piece of software.

3. It will generate the commands necessary to access data in each compenent system. The compiler will generate commands to follow certain access paths and unpack records. It will then translate these into the appropriate data accessing language or programming language to accomplish these tasks.

4. The compiler will deliver the requested information collocated, sorted and formatted to the user's specification. This specification will be a DIAM description of the output using the same algebraic access path theory, and necessary encoding details as are used to describe databases. The concept of using DIAM to describe formats of user presented data as well as databases was first discussed [6], but has not been possible to implement until now. This will allow the user to specify useful groups of data analogous to records, to group one type of data element hierarchically under another, and to prepare indicies. These capabilities are sufficient to create an input file for any application program that does not expect its input to be in the form of a network with pointers among blocks of data. Having specified such an output (of the compiler) or input (to an application), the user can then request the application to run on specified data by compiling a query with a particular selection criteria.

5. The compiler will plan how to make use of such temporary storage files as are necessary to answer the query and present it to the user in the form requested. It will generate the declarations and calling sequences necessary to make use of sorting and merging procedures available at the site(s) where the extraction of data is to be performed.

The first prototype compiler will not (i) model database systems at the level of physical storage allocation and seek times (as was done in the previous DIAMS software), (ii) implement telecommunications or on-line access automatically, (iii) model or resolve the issues of distributed procession in relation to telecommunications cost (as was done by Rothnie et. al. of C.C.A.), (iv) address issues of concurrency resolution, or (v) implement a friendly user front end.

# 7.0 FUTURE RESEARCH

7.1 The first prototype distributed database compiler will implement an algebraic language based on join, select, project and union. While this language is adequate to retrieve a super-set of any desired information, a more finely turned query language will require at least difference, count and sum. Exploiting the algebraic access path theory would add these operations to the query language and to the prototype compiler without excessive difficulty.

7.2 When information is missing, a query language should be capable of returning (i) those instances of an entity known to qualify, and (ii) those instances that could qualify if the missing information were known. It is important to (i) distinguish and define "unknown," "not-applicable," and "inconsistent." (ii) Extend the definition of relational algebra to include these, (iii) add additional algebraic operators required to exploit these, namely L-P (outer) join, outer union, and left and right outer joins, (iv) add these processing capabilities to the compiler.

7.3 The compiler as defined presently is for retrieval only. To complete the theory, one needs an algorithm for updates also. Unlike the updates of present relational languages, which operate on one data element at a time, such a request should be viewed as an instruction to add a structured file of data to a database. This task is to extned the language and compiler to accomplish this.

7.4 As a consequence of research on separate enterprise and data models, the next step should be to perform an experiment on several real databases; preparing a common semantic model of the ensemble, data models of each site, and exploring the process of translating from the semantic (entity) level to the data (relational) level. Semantic stability under perturbations of data schema will be examined.

# REFERENCES


1. Schneider, L.S. "A Relational View of the DIAM." in Proc. ACM SIGMOD International Conference on the Management of Data, Washington, D.C., June 1976, pp. 75-90.

2. Senko, M.E. "A Data Independent Architectural Model--Four Levels of Description." IBM Research, Yorktown Hts., N.Y., Report RJ982.

3. Codd, E.F. "Extending the Data Base Relational Model to Capture More Meaning." IBM Research, Yorktown Hts., N.Y., Report RJ2472.

4. Schneider, L.S. "A Relational Query Compiler for Distributed Heterogeneous Databases" in SHARE 50 Proceedings, Denver, CO., March 1978.

5. "Modelling for a Large-Scale Database--DIAMS Pacer Analysis." Prepared for RADC under contract F30602-77-C-0142, May 1978.

6. Schneider, L.S. and Spath, C.R. "A Generalized End-User Facility Architecture for Relational Database Systems" in Proc. Very Large Data Base Third Int. Conf., Tokyo, Japan, October 1977, pp. 359-369.

7. Bernstein, P.A. "What Does Boyce-Codd Normal Form Do?" Proceedings VLDB 1980, Montreal, Canada, pp. 245-259.

8. Schneider, L.S. and Spath, C.R. "Quantitative Data Description" in Proceedings of the 1975 ACM/SIGMOD International Conference on Management of Data, San Jose, CA., June 1975.

9. Codd, E.F. "A Relational Model of Data for Large Shared Data Banks," CACM Vol. 13, No. 6, June 1970.

VOLUME I

# The Separation of Enterprise

# and

# Data Models

# C O N T E N T S

## 1.0 SUMMARY

There is an ongoing debate between advocates of highly formatted models (e.g., relations) and more semantically oriented models (e.g., the entity set model) as to which achieves the best compromise between expressiveness at the enterprise level and mapability to the computer. The paper proposes that since these goals are essentially inverse duals, a compromise should not be attempted and in the approach presented, it is not. Instead, a two model system is proposed. One model depicts the enterprise independently of the data stored about it. The other describes the data with respect to the enterprise. Finally, a mappping between the models is shown not only to support query compilation, but also to clearly define what data does or does not exist with respect to the enterprise and its precise meaning. The result is a modelling system substantially more capable of integrating amorphous data than any models yet developed.

## 2.0 INTRODUCTION

This paper presents an approach to integrated modelling of information in heterogeneous databases. It differs from other approaches in that both an enterprise model and a data model are employed and these are fully separated from each other.

The enterprise model identifies the entities and relationships that are pertinent to the enterprise. Its most unusual feature is that it models entities that are not names (e.g. people) without requiring any simple method of identification. While most databases are designed to force unique identifiers on important entities, this is not done in any consistent manner from one database to the next. The only realistic assumption for integrating heterogeneous databases is that identification is a problem which may be only partially solvable.

The data model identifies the names and associations among names that compromise a database or collection of databases. We find the conventional relational model to be fully adequate for modelling data. On the other hand, the relational model has been criticized by exponents of more semantically oriented models for confusing issues of meaning. In our method, this is not a problem because the data model is not required to perform a semantic function.

The issue of mapping down or "compiling" is of great importance for this approach to be practical. The data model is defined in terms of the enterprise model which not only provides meaning for the data model but provides a basis for compiling queries expressed at the enterprise level into queries at the relational level. This piece of database architecture is consistent with the relational DIAM [1,2] approach of compiling of relational queries into access path dependent queries for the purpose of distributed heterogeneous database access.

The primary advantage of performing integration at the enterprise level is that it is more stable. The relational model cannot generally overcome the semantic differences that occur between administrative centers. This paper addresses the issue of how to maintain query stability under changes to the enterprise schema, using methods that are not available at the relational level.

This method uses the separated enterprise model to apply careful attention to semantics. For this we are indebted to Michael Senko and William Kent. The mathematics required for the various languages and mappings is not novel; it is first order logic which is the foundation for all relational languages already. In this paper, mathematical and linguistic considerations have been relegated to the background to emphasize the dual model approach.

## 3.0 THE CONTEXT OF SEMANTIC MODELLING - DATABASE INTEGRATION

The dual model approach described here was developed as part of an overall research effort to integrate heterogeneous data bases. Our goal is to put a common umbrella over a diverse collection of existing databases having various implementations so that a user can query the collection as if it were a single integrated system without having to know where or how the data is stored.

The original proposal [2] was to create a common relational model for the collection of databases and then to define mappings from the relational model into the access methods of the various databases. There is no problem in principle with creating a relational model of an existing database whose explicit schema is given as a network, hierarchy or other data description technique. However, to make a good relational model (i.e., one that obeys the various normal forms [3,4] and is "well defined" [5]) one must know more about the enterprise than can be gleaned from DDL schemas or data dictionaries. Our enterprise model provides a systematic way for recording this information.

A map from a relational model to a database implementation is a compiler that accepts transactions expressed in a relational language and produces programs that access data using whatever data manipulation facilities are provided. This could theoretically

take place at any level. One could conceive of a database with no accessing software, in which case the compiler referred to above would have to generate programs to read streams of data on storage devices. In practice, it might be generating programs that use of the DML facilities of, say, a CODASYL type language or IMS. The technology for this compiler is described in [2]. It makes use of DIAM (Data Independent Accessing Model) [1,6,7] to describe implementations. The compiler is general in the sense that DIAM is capable of describing almost all known storage structures because of the principles of storage which it recognizes, and the compiler, when given a DIAM description of a database, can then compile accesses into it. This technology has been given a pilot test [8] on a complicated real database which indicated the viability of creating a relational model of non-relational database and of using a DIAM compiler to generate good accessing programs for queries.

To complete this project, one must have a practical method of creating a common relational model for a collection of databases. This is more complicated than having a relational model for each separate database, and requires considerable semantic knowledge of the integrated enterprise or environment in which all the databases reside. Many authors [9,10,11,12,13] have commented on the semantic limitations of the relational model. When one is attempting inte-

gration rather than designing a new system, these problems are intensified and new ones appear.

Database administrators have the opportunity to create unique identifiers when they feel these are needed. The integrator of databases is in a more passive role; he has to take what is given. This may include the same people recognized by different identifiers in different databases. Therefore, we have rejected as unrealistic the assumption that each entity set have a primary identifier. We expect partial and overlapping identifiers will exist and must be modeled.

Another problem is that of generalization. What does one mean by "truck" or "motor vehicle?" Even assuming that everyone agrees that all trucks are motor vehicles, the distinction between passenger cars and trucks may differ from state to state and be based on specified attributes. One state's truck may be a union or restrictions of another state's trucks, cars and recreational vehicles. Whatever definition of "truck" is offered to a user must be adequately explained, and appropriately translated for uniform access.

The recorded attributes of an entity may vary from database to database. One database may record ownership records and liens on trucks and another may record repair histories of an overlapping set of trucks. Or the data recorded for trucks may be different than the data recorded for cars.

The correspondence between relationships among entities in the enterprise model and relations (or projections of relations) among names in the data model is often not one-to-one. As the examples in this paper show, this is sometimes an essential situation that is not merely the result of poor choice of a relational schema.

These problems have no simple solutions. They have to be worked out in detail by the administrator of the database integration. It is possible, however, to have a simple common language for performing this integration. It can consist of just the relational model,[1] and a language having about the same expressive power as relational algebra or calculus.

The important point is one of general semantics [14]. Name and object are not the same, and careful attention must be paid to each one and to the connection between them. Therefore, we have the two models, and a mapping from the enterprise model to the data model.

The enterprise model is one of people, vehicles, etc., not their names or even their "surrogates" [15]. Database theoreticians have generally avoided this approach because they correctly realize that a database can only process names. But this restriction is not as severe as one might think. Since language possesses generic as well as individual names, there is no problem with mak-

[1] Both of our models could be called relational, although they differ in what they model.

ing modelling assertions about entities in general. When we realize this, it turns out that we already have the modelling capability necessary to overcome many problems of naming within databases by referring to the underlying entities.

## 4.0 THE ENTERPRISE MODEL - BASIC DEFINITIONS

An enterprise schema makes use of the following concepts: Entity sets, name sets, relationships and identity maps.

Entities are the perceived independent objects of the enterprise that are noteworthy. Entities may be people, physical objects, ideas, events, actions, qualities, properties, values, etc. This model does not distinguish "attributes" from other types of entities.

Entity sets are sets of entities using the standard mathematical notion of "set". In general, an entity may belong to more than one entity set. Thus the standard mathematical concepts of union, intersection, difference, subset, cover and partition may apply to the discussion of certain situations. Our intention is to create a first order model only, and therefore we do not allow an entity set to be a member of an entity set, although an entity set may be a subset of another entity set. If in the future we wish to propose a second order language over the entity sets, then sets of entity sets will be called families.

Names are those entities which consist of strings of characters of some alphabet or character set and can be stored in databases, printed on paper and displayed on terminals. Names of people, social security numbers, numerical values, names of paint

colors, etc. are names. People, airplanes and congressional districts are not. A database is of necessity a collection of names in association with each other according to some schema.

Name sets are entity sets comprised of names only.

A relationship between two entity sets A and B is a set of ordered pairs, the first of each pair being a member of A and the second being a member of B. Thus every relationship has associated with it two entity sets which we shall call the subject and the object respectively. The subject and object of a relationship may obviously be the same entity set, and there may be more than one relationship between any pair of entity sets. Relationships are dependent objects of the model in contradistinction to entity sets which are independent objects because relationships cannot exist without entity sets (by fiat because of our definition.)

A certain amount of casualness in the definition of names is often tolerated in data processing. In general one does not get into trouble by confounding a thing and its name if (i) these are in strictly one-to-one correspondence; and (ii) one does not confuse attributes of an entity (the stripes on a tiger) with attributes of its name (the "t" in "tiger".) As a practical example, one may simplify a model by treating the color of a car as a symbol if no distinction between a color and its name will ever have to be made, and

the names are in one-to-one correspondence with the colors. In this paper, it is not necessary to define where one should draw the line.

An identity map is a relationship each of whose ordered pairs contains the same entity as both subject and object. Identity maps permit referencing the same entity as a member of two or more different entity sets. They are used to model generalization, among other things. One might think that an identity map is redundant informationally. If the subject and object were both names, this would be true since the identity could be computed by comparing them letter for letter. But we intend identity maps to be used on non-name entity sets. It may be non-trivial to identify the person with social security number 234-55-3056 as the same as the person with driver's license K119145.

Relationships should be understood to include identity maps as a special case.

In order to specify an enterprise schema, the entity sets and relationships must be given names. An enterprise schema then consists of:

i) A list of entity set names,

ii) Indication of which entity sets are name sets,

iii) A list of relationship names, together with a subject and object for each relationship,

iv) Indication of which relationships are identity maps, and

v) Optionally, inverse names for relationships. These are the names for the relationships with subject and object reversed.

Another way to present a schema is pictorially. Enterprise schema can be viewed as graphs with entity sets as nodes and relationships as edges. We shall adopt the convention of using diamonds for name sets, ovals for other entity sets, dashed lines for identity maps and solid lines for other relationships. The name of an entity set appears inside the diamonds or oval. The name of a relationship appears along the line together with an arrowhead that points from subject to object. If there is an inverse name, it appears along the same line but with the arrowhead pointing in the opposite direction. Such a diagram contains exactly the same information as a schema presented as a listing of (i) through (v) above.

## 5.0 THE ENTERPRISE MODEL AND REALTY

In creating the enterprise model, we have tried to make only the minimum necessary assumptions about the nature of an enterprise. An enterprise is that which is perceived to be real and of interest to a group of people involved in some endeavor. Perception occurs prior to modelling; how a group concensus is obtained and what constitutes reality is beyond the scope of this paper. The modelling technique must merely be general enough to record the results.

At the minimum, entities and some categorization of entities are perceived and the categories overlap. Some perceptions will be be completely unclassifiable (i.e., unique) but we do not expect a database to admit these as data; i.e., a database must make some distinction between schema and instance to be manageable.

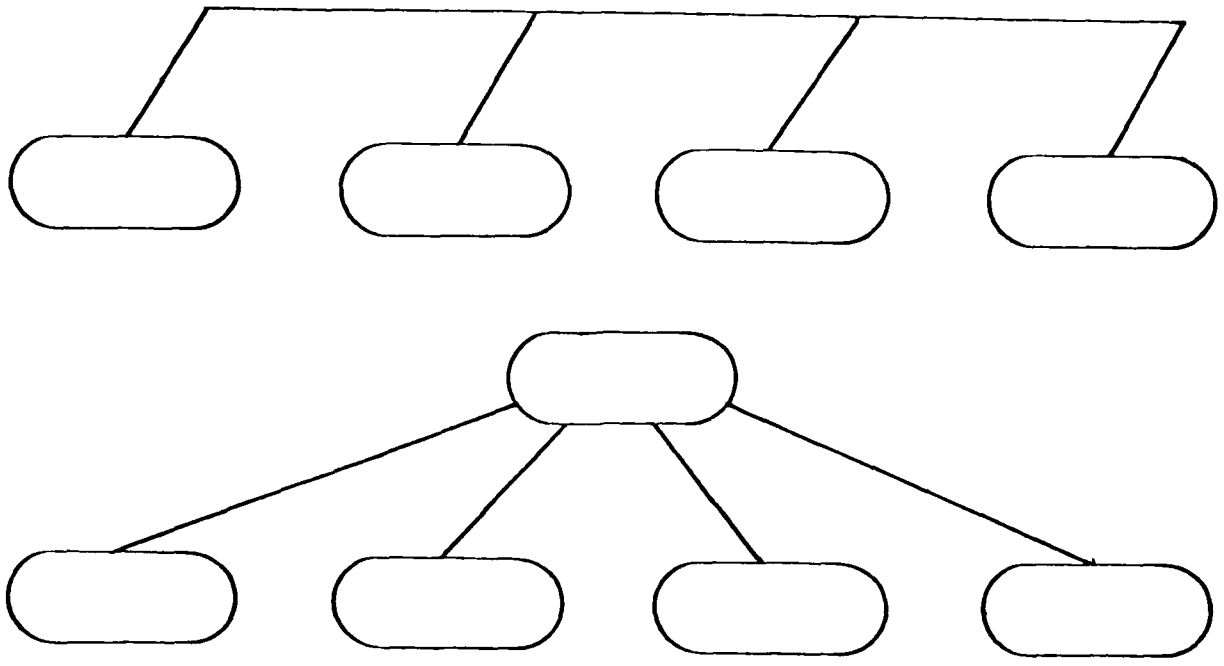
We do not believe that there can be any consensus for distinguishing entities, facts, attributes, values, properties, relationships, etc. Why not have just one type of primitive object in the model? We do not know any genuine way to do this, so we have two primitive notions, entities and relationship instances. It seems that we need some things to begin with, and then some more things which are about or between the first ones. However, the reality of the situation itself is not always going to make this distinction for us in any unambiguous way. It probably depends on what we

choose to notice first. An entity, in our model, has independent existence and may relate in various ways to other entities. A relationship instance exists only as a connection between two entities and has no other properties. But as one pays more attention to a relationship, it may appear to have an independent existence and attributes of itself worth recognizing. The model provides for an orderly way for the schema to grow so as to accommodate this new perception, while preserving the stability of existing transactions.

This model uses only binary relations.[1] This is purely a convention, and we do not claim that relationships as perceived are necessarily binary. But it is not a restrictive convention. An n-ary relation can always be modeled as an entity set and n binary relations. (See Figure 1).

Having one entity set and n binary relations in the schema requires exactly as many names as having one n-ary relation and n role-names [16]. The advantage of this convention is the graph structure which simplifies diagramming of the model.

[1] The data model uses n-ary relations. We shall comment on binaries and n-aries later.



AN N-ARY RELATION vs. AN ADDITIONAL ENTITY SET
AND
N BINARY RELATIONS

Figure 1

# 6.0 THE ENTERPRISE MODEL - EXAMPLES

Example 1:

A corporation assigns employee numbers to employees. The personnel office is interested in keeping track of these numbers, the names of employees and their spouses.

The enterprise model can be derived from the following assertions:

1) Employees are people.

2) Some employees are married. This is a relationship to another person.

3) People have names.

4) People have birthdays.

5) A birthday is a day.

6) Days can be named by dates.[1]

7) Employees are assigned employee numbers.

An enterprise model is now presented both as a listing in Figure 2 and as a diagram in Figure 3. After this example, only diagrams will be used.

Example 2:

The Social Security Administration assigns social security numbers to participants as illustrated in Figure 4.

---

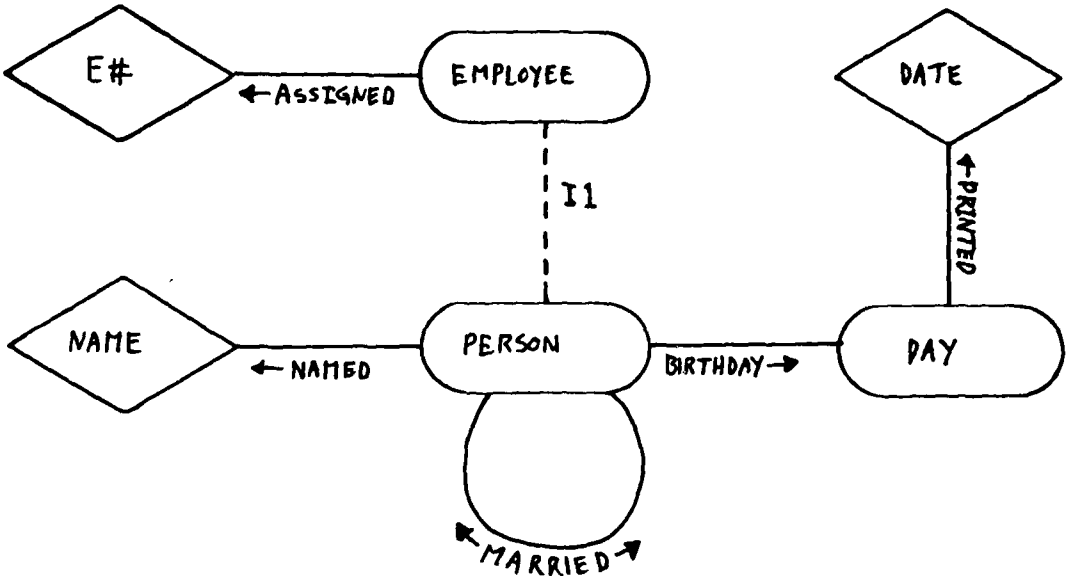
[1] Days can be referenced in other ways also, e.g., "the day after tomorrow," "George Washington's Birthday," or by means of the Mayan calendar.

| | | |
|---|---|---|
| ENTITY SETS: | EMPLOYEE | |
| | PERSON | |
| | DAY | |
| | E# | (Name) |
| | NAME | (Name) |
| | DATE | (Name) |

| RELATIONSHIPS: | SUBJECT | OBJECT | INVERSE | |
|---|---|---|---|---|
| NAMED | PERSON | NAME | | |
| ASSIGNED | EMPLOYEE | E# | | |
| BIRTHDAY | PERSON | DAY | | |
| MARRIED | PERSON | PERSON | MARRIED | |
| PRINTED | DAY | DATE | | |
| I1 | EMPLOYEE | PERSON | | (identity) |

THE ENTERPRISE MODEL (Listing)

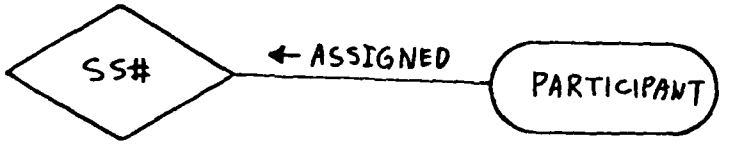
Figure 2



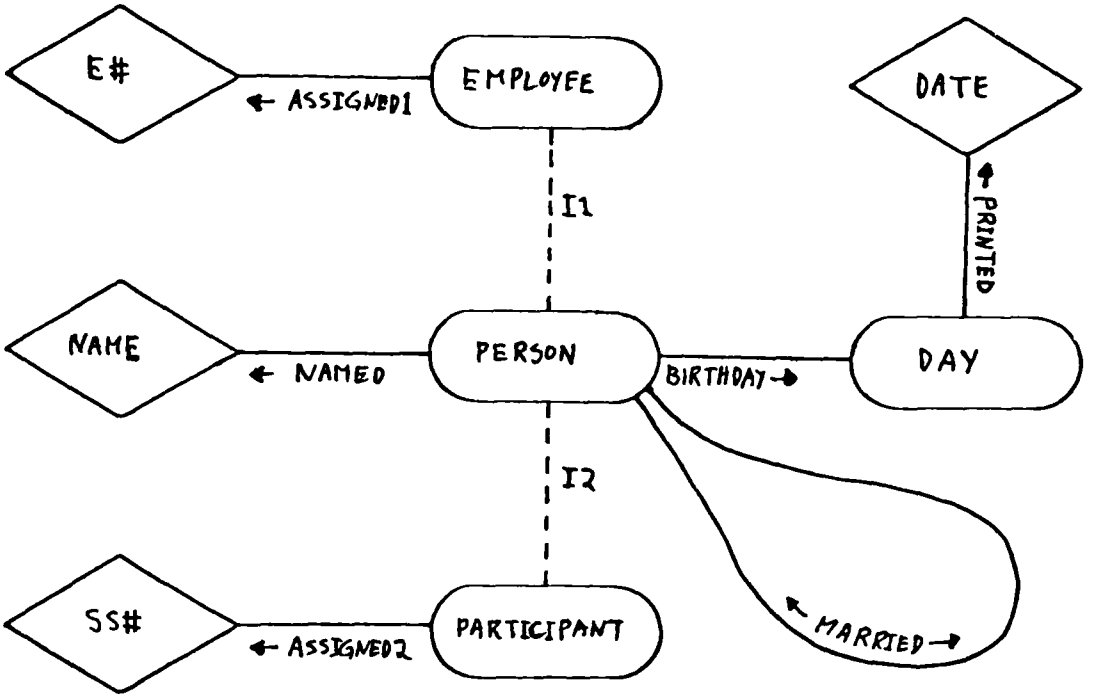

THE ENTERPRISE MODEL (Diagram)
Figure 3

Example 3:

Integration of examples 1 and 2 results in Figure 5. Any group of schemata can be unified although some renaming and additional assumptions may be necessary. When they involve some common concepts, the integrated model will be connected.



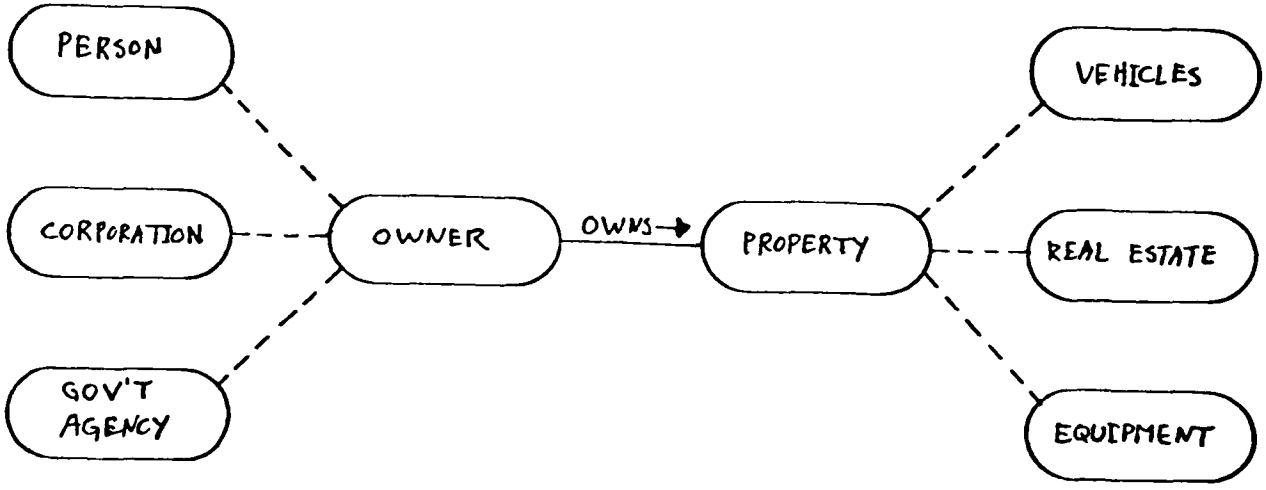

SOCIAL SECURITY ADMINISTRATION ENTERPRISE
Figure 4




INTEGRATION OF EXAMPLES 1 AND 2
Figure 5

Example 4:

Persons, corporations, and government agencies may be owners of vehicles, real estate and equipment. This is illustrated in Figure 6.




OWNERS OF PROPERTY

Figure 6

## 7.0 THE ENTERPRISE MODEL - MAPPING ASSERTIONS

The aspect of the model introduced so far specifies only the terminology and connectivity of the enterprise. There is a great deal more that can be said about an enterprise. Of particular importance are the mapping properties of the relationships. These must be known in order to design data models and storage structures, and to define the compilations.

Functional dependencies [3,17,18] are one of the most important of mapping properties. The object of a relationship is functionally dependent on the subject (via the relationship) when there is at most one object for each subject. More generally, a collection of relationships having the same object express a functional dependency of the object on the collection of subjects when each ensemble of subjects has at most one object associated via the collection of relationships. Functional dependencies have a useful transitive property (A,B->C and D,E->F and C,F->G imply A,B,C,D->G) which makes unique naming systems possible.

Stronger than the functional dependency is the assertion that for each subject there is exactly one object. This creates a total function. More generally, we can have assertions that for each subject there are at least m and at most n objects.

The inverse of a relationship has its own mapping assertions which are not necessarily the same as those of the original relationship. For example, there may be a functional dependency in one direction, but not in the other direction.

Mapping assertions on identity maps have a somewhat different significance. Identity maps are necessarily one-to-one in both directions, so this does not need to be repeated. If an identity map from A to B is a total function, then A must be a subset of B. Given a collection of identity maps with a common object, this may be an instance of a cover, or a partition (which is a stronger assertion). These assertions on identity maps are the tools necessary for dealing with generalization [5].

Example 5:

The assertions of Figure 7 are valid for the enterprise model of Figure 3.

| Relationship | Bounds on Number of Objects |
|---|---|
| ASSIGNED | EXACTLY 1 |
| ASSIGNED INVERSE | AT MOST 1 |
| I1 (FROM EMPLOYEE TO PERSON) | EXACTLY 1 |
| NAMED | EXACTLY 1 |
| MARRIED | AT MOST 1 |
| BIRTHDAY | EXACTLY 1 |
| PRINTED | EXACTLY 1 |
| PRINTED INVERSE | EXACTLY 1 |

ASSERTIONS OVER FIGURE 3

Figure 7

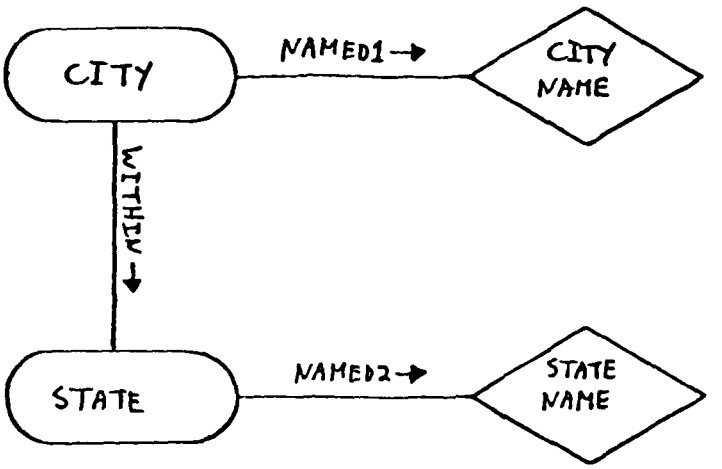
Example 6:

Cities are uniquely identified by the name of the city and the state, which in turn is identified by its name. (See Figure 8).

From these assertions, one can deduce that CITY NAME and STATE NAME are functions of CITY, and that CITY is functionally dependent on CITY NAME and STATE NAME. These conditions jointly determine a complete unambiguous naming system for CITY since CITY NAME and STATE NAME are name sets. This is not always the case. If the functional dependency does not exist, then a name system is ambiguous as for example the name of a person. If the total function does not exist in the opposite direction, then the naming system is incomplete.

## 8.0 THE DATA MODEL

The data model used here is the relational model expounded by Codd [19], i.e., an n-ary model over domains which are name sets. The purpose of a data model is to define the contents of a database (or ensemble of databases) in a representation-independent manner. A database is meaningful only because it makes assertions about an enterprise; therefore the definition of the data model in terms of the enterprise model is important because it allows us to interpret the presence or absence of tuples (relational instances) as being meaningful assertions about the enterprise.




| Relationship | Bounds |
|---|---|
| NAMED1 | EXACTLY 1 |
| NAMED2 | EXACTLY 1 |
| NAMED2 INVERSE | EXACTLY 1 |
| WITHIN | EXACTLY 1 |
| WITHIN INVERSE, NAMED1 INVERSE | MOST 1 |

NAMING OF CITIES AND STATES
Figure 8

Example 7:

The personnel office (see Example 1) has a database which is modeled by the relational schema of Figure 9. The relation EMP is in fourth normal form. The relationship BIRTHDAY of the enterprise model is expressed in two different ways here for employees' birthdays and employees' spouses' birthdays. There is no reasonable way to simplify this. Any person's birthday is functionally dependent on that person, but not necessarily functionally dependent on that person's non-unique name. Note also that the spouse's name is functionally dependent on E# only in a monogamous society which was asserted in Figure 7. Otherwise it is a multi-valued dependency.

Relation: EMP

| Attributes: (role-name) | (domain) |
|---|---|
| E# | E# |
| NAME | NAME |
| BIRTHDAY | DATE |
| SPOUSE | NAME |
| SPOUSE-BIRTHDAY | DATE |

Key: E#

DATA MODEL FOR EXAMPLE 7

Figure 9

The data model can be defined with respect to the enterprise model as follows (Figure 10):

```
FOR-ALL E# X, NAME Y, DATE Z, NAME V, DATE W
  EMP (X,Y,Z,V,W) IMPLIES
    THERE-EXISTS EMPLOYEE A, PERSON B, DAY C
      WHERE (A ASSIGNED X) AND (A Il B) AND
        (B NAMED Y) AND (B BIRTHDAY C) AND
          (C PRINTED Z) AND
            V ≠ NULL IMPLIES
              THERE-EXISTS PERSON D, DAY E
                WHERE (B MARRIED D) AND (D NAMED V)
                  AND (D BIRTHDAY E) AND (E PRINTED W)
```

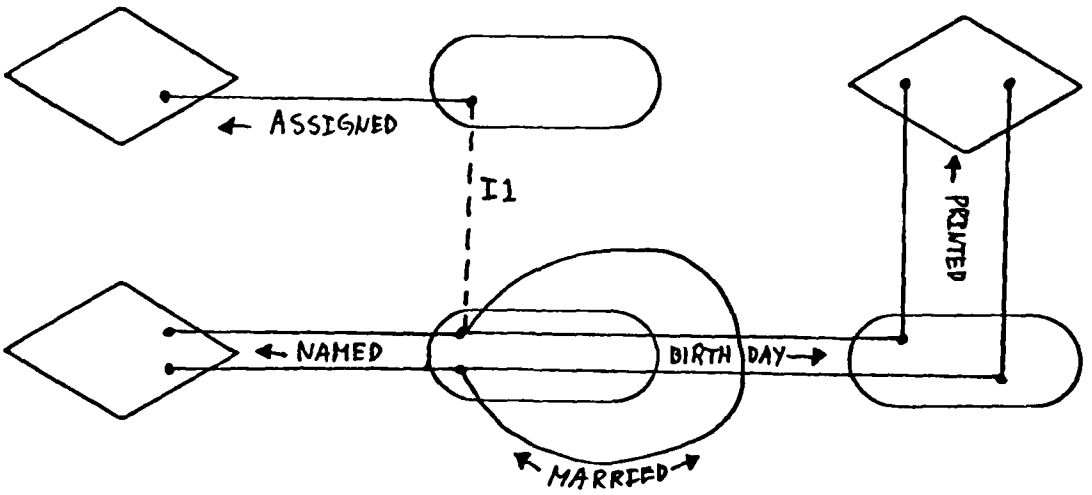
DEFINITION OF EMP RELATION

Figure 10

Notice that we have asserted that the existence of a tuple in the database implies the existence of an employee but not visa versa. Also, we have allowed for a null value in the SPOUSE column to imply nothing, but any other value must be the name of the spouse, which requires that the spouse's birthday also be supplied. These are not the only possibilities. We may wish to assert that the non-existence of a tuple in the database implies the non-existence of an entity having the name. We may allow several special

values in any domain signifying non-applicable, nonexistent, existent but unknown, etc. Discussion of the algebraic manipulation of "unknown" can be found in [13] and [20].

In many cases, a definition of a single relation can be broken down into a disjunction of cases (e.g., with spouse and without spouse) such that each case can be represented by a line diagram over the enterprise model. The semantics of the EMP relation where there is a spouse is shown in Figure 11.




EMP RELATION SUPERIMPOSED ON FIGURE 3
Figure 11

## 9.0 STABILITY OF THE ENTERPRISE MODEL

Growth of an enterprise schema can be broken down into a series of small steps. These steps are of five kinds, and in each case we show how to maintain stability of existing queries by making the old schema appear and as a transformation of a new one. Such transformation may also be used to provide particular users a partial view of the database.

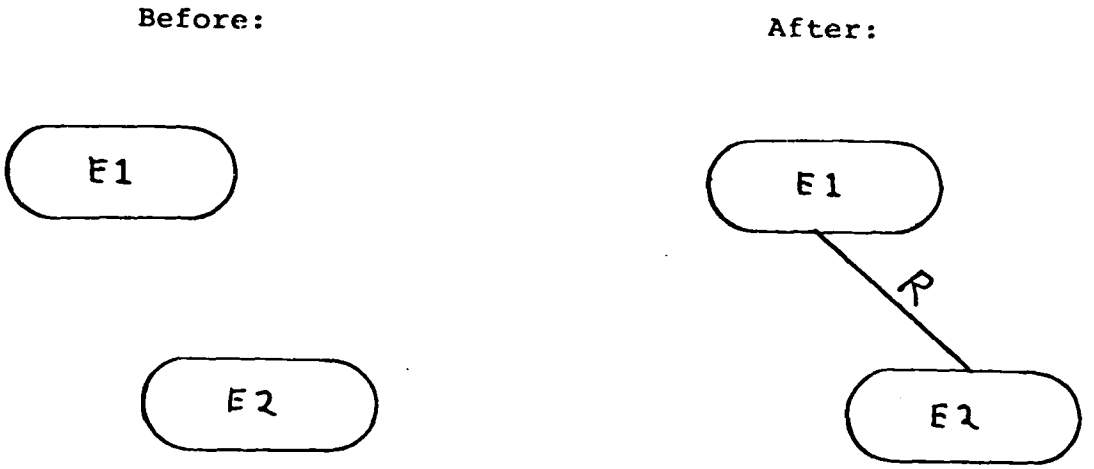
Type 1 step:

A new entity set appears, as yet not related to the model. No change is required to existing queries. (See Figure 12).


TYPE 1 STEP
Figure 12

Type 2 step:

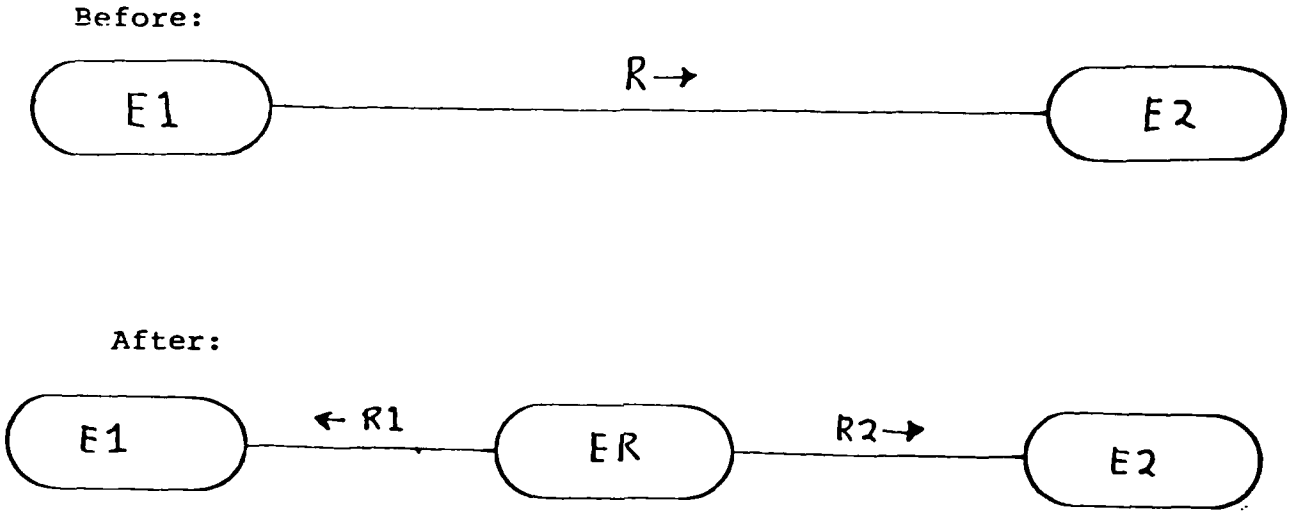
A new relationship appears between existing entity sets. No change is required to existing queries, although some new queries are now possible. (See Figure 13).




TYPE 2 STEP
Figure 13

Type 3 step:

A relationship is transformed into an entity and two new relationships appear (See Figure 14).



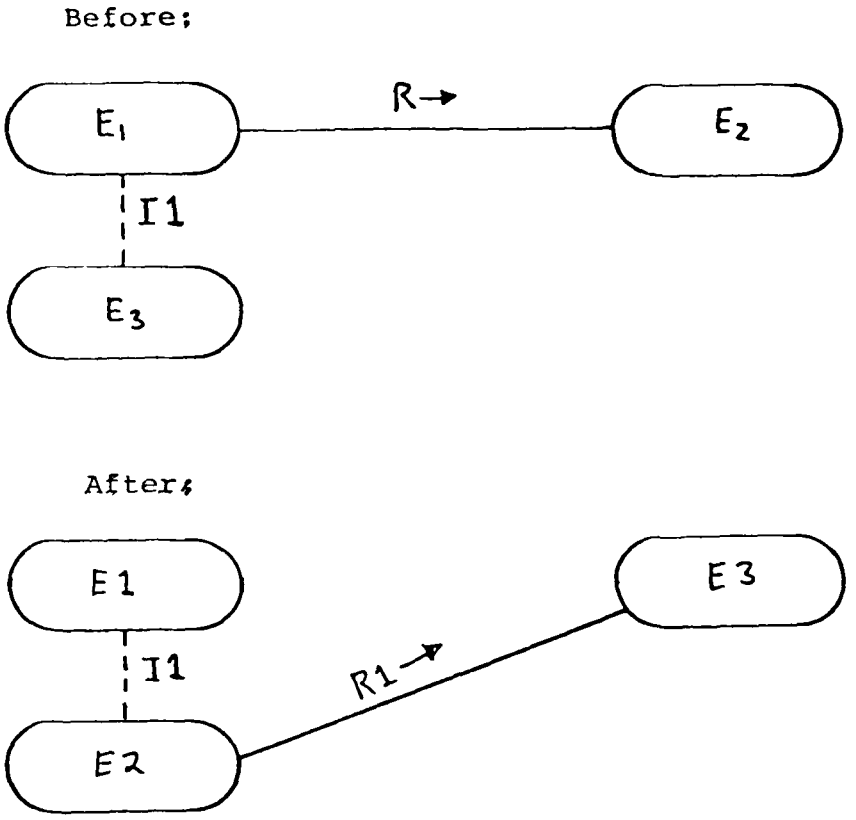

TYPE 3 STEP

Figure 14

Queries containing the relationship R must be redefined. The expression (X R Y) must be replaced by (THERE-EXISTS ER Z WHERE (Z R1 X) AND (Z R2 Y)). Algebraically, this is the natural join of R1 and R2.

Type 4 step:

The path of a relationship is redefined by inclusion of an identity map. (See Figure 15).



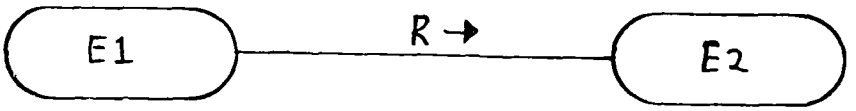

TYPE 4 STEP
Figure 15

Queries containing R must be redefined. The expressin (X R Y) must be replaced by (THERE-EXISTS E3 Z WHERE (X I1 Z) AND (Z R1 Y)). Algebraically, this is the natural join of I1 and R1.

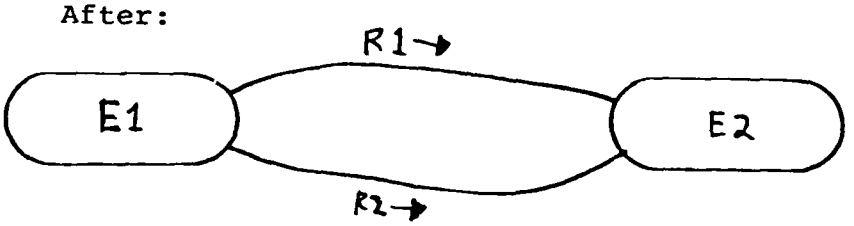
Type 5 step:

A relationship is partitioned into several cases. (See Figure 16).

Before.




After:

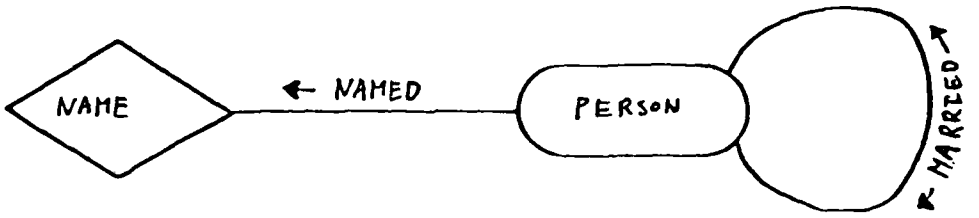



TYPE 5 STEP
Figure 16

The expression (E1 R E2) must be replaced by ((E1 R1 E2) OR (E1 R2 E2)). Algebraically, R is the union of R1 and R2.

Example 8:

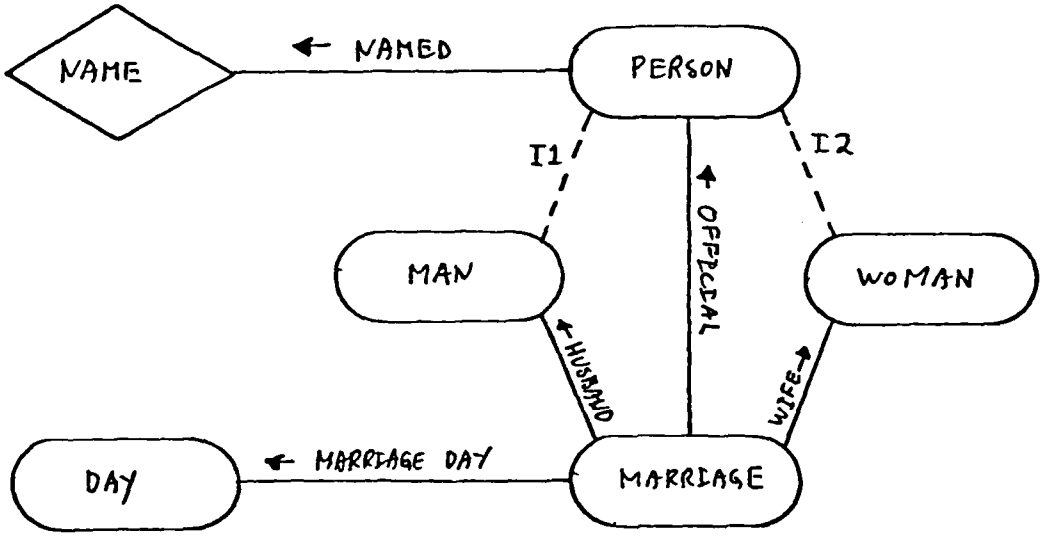
Consider change to the schema in Figure 17.




SCHEMA FOR MARRIED

Figure 17

The model is to be refined by partitioning PERSON into MAN and WOMAN, and allowing marriage to become an entity having a date and an official who presides at the ceremony. (See Figure 18).




SCHEMA FOR MARRIAGE

Figure 18

This can be accomplished by a sequence of changes of types 1 through 5. The expression (X MARRIED Y) must be replaced by:

```
(THERE-EXISTS MAN V, WOMAN W, MARRIAGE X WHERE
 (Z HUSBAND V) AND (Z WIFE W) AND (((X I1 V)
 AND (Y I2 W)) OR ((X I2 W) AND (Y I1 V)))).
```

The purpose of these transformations is to allow a query expressed over an old schema to retain its semantics by mapping it into a query over the new schema. Provided that the new schema represents growth rather than shrinkage of the database, and provided that whenever data is merged, an attribute is added to indicate where it came from, this appears to be possible.

## 10.0 QUERY COMPILATION

Given an enterprise schema, one may formulate a query using some first order query language. Such a query must have a target list of variables that range over name sets rather than any other entity sets if it is to be processed by a database system rather than a vending machine. But the quantified variables occuring within the query may range over non-name entity sets.

For example, one may ask a database system to get the name of the spouse of employee having E# = 01438, but one may not ask for the spouse of that employee. The former query may be expressed as:[1]

```
GET NAME X WHERE
  THERE-EXISTS PERSON A, PERSON B, EMPLOYEE C AND
    (A NAMED X) AND (A MARRIED B) AND
      (B I1 C) AND (C ASSIGNED "01438")
```

The variables A, B and C are over people but are not input or output to the query. One can readily see that this query is answered by the following database query (See Example 7):

```
GET EMP/SPOUSE WHERE EMP/E# = "01438"
```

The purpose of compilation is to produce this transformation automatically by using the definition of the data model.

---

[1] See Appendix C on Language

There are a number of complications that can occur, one of which we would like to illustrate -- the resolution of identities. Using the enterprise model of Figure 5, suppose that an integrated database schema for Corporation X and the Social Security Administration is given in Figure 19.

Relation: EMP

| Attributes: | (Role-Name) | (Domain) |
|---|---|---|
| | E# | E# |
| | SS# | SS# |
| | NAME | NAME |
| | SPOUSE-NAME | NAME |
| Key: | E# | |
| Alt. Key: | SS# | |

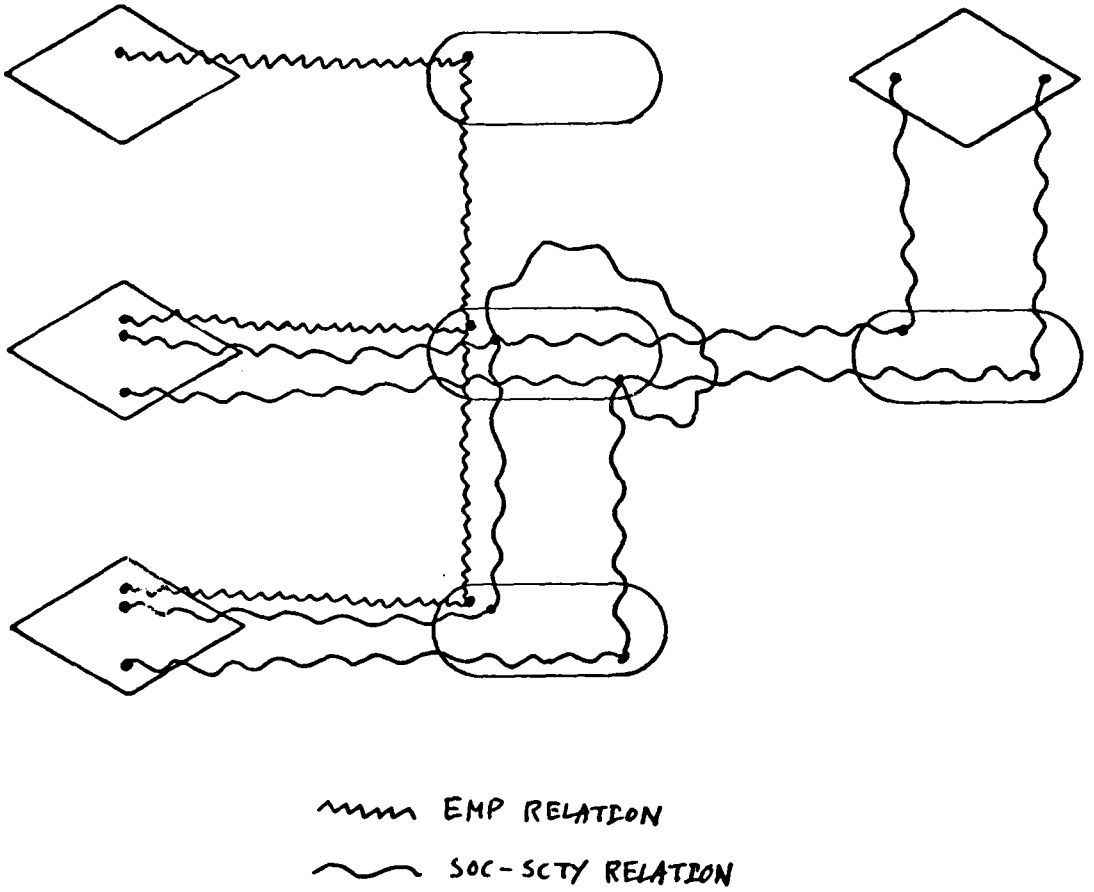
Relation: SOC-SCTY

| Attributes: | (Role-Name) | (Domain) |
|---|---|---|
| | SS# | SS# |
| | NAME | NAME |
| | BIRTHDAY | DATE |
| | SPOUSE-SS# | SS# |
| | SPOUSE-NAME | NAME |
| | SPOUSE-BIRTHDAY | DATE |
| Key: | SS# | |

DATA SCHEMA FOR INTEGRATED DATABASE

Figure 19

In Figure 20, we have drawn both these relations as paths over the enterprise model.

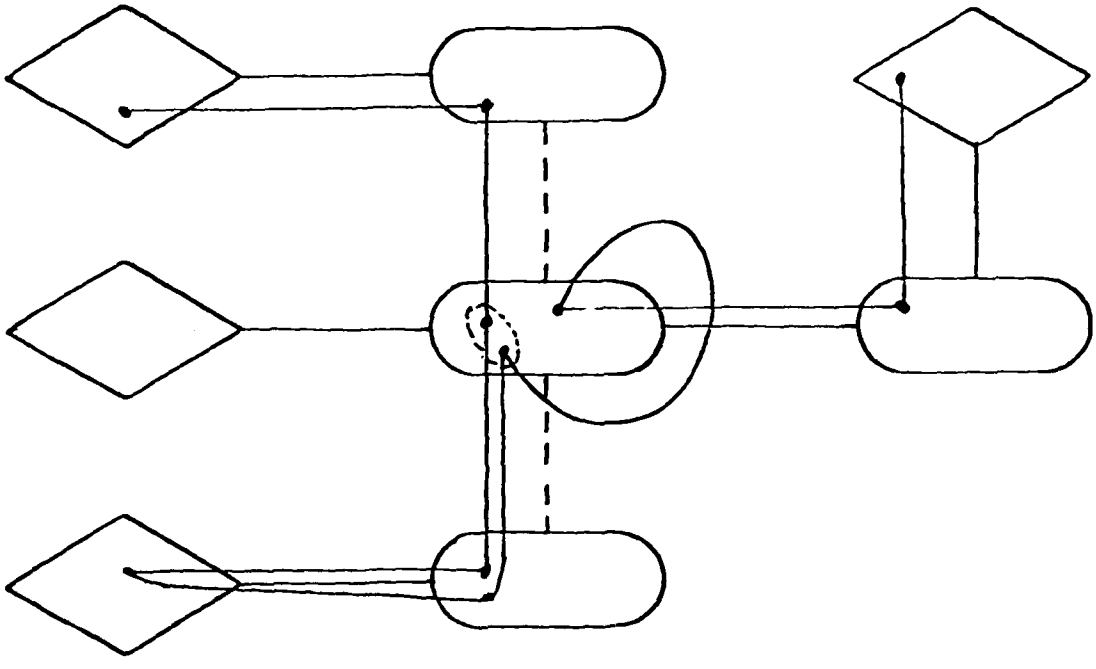



TWO DATA RELATIONS SUPERIMPOSED ON AN ENTERPRISE MODEL
Figure 20

Consider the Query:

```
GET DATE D WHERE
  THERE-EXISTS EMPLOYEE X, PERSON Y, DAY Z AND
    (X ASSIGNED "01438") AND (X MARRIED Y) AND
      (Y BIRTHDAY Z) AND (Z PRINTED D)
```

This query specifies an employee (X) married to a person. So it is necessary to identify the employee as a person using I1. Since the EMP relation of this example does not have the spouse's birthday, a join must be made to the SOC-SCTY relation. Mapping assertions should make it clear that the only non-ambigious join is on SS#, not NAME. The query with identifiers resolved is shown in Figure 21.



QUERY WITH IDENTIFICATION RESOLVED

Figure 21

Finally, the query is resolved to the level of relations:

```
GET SOC-SCTY/SPOUSE-BIRTHDAY WHERE
  THERE-EXISTS EMP AND EMP/E# = "01438" AND
    EMP/SS# = SOC-SCTY/E#
```

Our investigation of compilation is currently centered on (i) the amount of assertion capability required to define the data model in terms of the enterprose model over a wide range of realistic situations, and (ii) the compiler technology required to transform enterprise queries into data queries.

# 11.0 CONCLUSION

We have shown that a clear separation between an enterprise model and a data model allows for better modelling of both. The essence of this technique is the fact that the enterprise model makes it clear what facts about the enterprise model are or are not expressed by the data, and if so, how they are expressed.

# APPENDIX A - FUNCTIONAL AND MULTI-VALUED DEPENDENCIES

The study of functional and multi-valued dependencies and normal forms is generally an attempt to apply a mathematical discipline to relational theory so that it mirrors good semantics. As we have made clear, an understanding of functional dependencies at the level of the enterprise is necessary to handle problems of naming and identification correctly.

As the relational theoreticians pursued the subject or normalization further, they found it necessary to recognize the more general multivalued dependency. This is interesting because at the enterprise level, binary relationship and paths through the model (which are natural join sequences of relationships) are always multi-valued dependencies.

If one designs enterprise models before data models, then the process of designing the data model appears to be:

i) Discovering naming systems for important entities.

ii) Creating relations that are existence lists for these entities. These relations are all key.

iii) Creating relations that express a single binary relationship or join sequence of relationships.

iv) Combining several of the above because they have a common subject and all the objects are functionally (not multi-valued) dependent on the subject, therefore the naming system of the subject is the key of the relation.

The most common disruption to a data model (and to a storage structure) occurs when a functional dependency becomes multi-valued. (e.g., Corporation X establishes a branch in a country where polygamy is practiced.) This enlarges the key of those relations defined in step (iii) and requires the factoring of relations that have been combined in step (iv). As noted, the topology of the enterprise model is not affected by this type of change because the underlying semantics have not been disturbed. The same relationships exist, whether single valued or multi-valued.

# APPENDIX B - BINARY vs. N-ARY RELATION

There has been much heated argument on this subject. The subject can be clarified by asking whether one wants to model the enterprise or the data.

A model of associations of (atomic) names stored in a database is of necessity n-ary. A ternary relation with a multi-attribute key in fourth normal form cannot be factored into projections without loss of information.

Those who advocate binary models, such as Senko [6,9], generally wish to emphasize the entities of the enterprise rather than their keys. This makes good sense semantically. But if one neglects the issue of naming systems or considers only the case of a complete unambiguous naming system, one has a weaker system than a relational system which can always side step semantic issues to get the job done.

## APPENDIX C - LANGUAGE

This paper does not present a language proposal and the various examples should not be construed as such. The question of how powerful a transformation is needed to compile from the enterprise to the data level has not been answered, and can only be answered by extensive investigation. All the examples contained in this paper are well within the expressive bounds of relationally complete languages [1,2]. In fact, they can all be expressed in a weaker language without universal quantifiers which we have called "existentially complete." This is equivalent to a relational algebra lacking difference and division.

# APPENDIX D - GENERAL SEMANTICS

The basic principles of General Semantics [14,22] are (i) the map is not the territory, (ii) the map never completely represents the territory, and (iii) a map of a map is a map of a different order.

We have been influenced by principle (i) to make two separate models, and by principle (ii) to allow for continual growth of the modelling process. We have respected principle (iii) to the extent of not confounding maps and meta-maps. We have only considered the former. But eventually, maps of maps will be needed. When the schema of an enterprise or database becomes large enough, it itself must be treated as a database, as in [13]. We are also investigating this idea.

## REFERENCES

1. Schneider, L.S. "A Relational View of the Diam." Proc. ACM SIGMOD International Conference on the Management of Data, Washington, D.C., June 1976, pp. 75-90.

2. Schneider, L.S. "A Relational Query Compiler for Distributed Heterogeneous Databases." SHARE 50 Proceedings, Denver, Colo. March 1978.

3. Chamberlin, D. D. "Relational Data Base Management: A Survey." Computer Surveys, Vol. 8, No.1, March 1976.

4. Fagin, R. "Multi-valued Dependencies and a New Normal Form For Relational Database." Vol. 2, No. 3, September 1977, PP. 262-278.

5. Smith, J.M. and Smith, D.C.P. "Database Abstractions: Aggregation." Comm ACM, Vol. 20, No. 6, June 1977.

6. Senko, M.E. "A Data Independent Architectural Model -- Four Levels of Description." IBM Research, Yorktown Hts., N.Y., Report RJ982.

7. Levin, M. "An Introduction to DIAM - Levels of Abstraction in Accessing Information." ACM '78, Washington, D.C., December 1978.

8. "Modelling for a Large-Scale Database -- DIAMS Pacer Analysis." Prepared for R.A.D.C. under contract F30602-77-C-0142, May 1978.

9. Senko, M.E. "DIAM II: The Binary Infological Level and Its Database Language -- FORAL." IBM Research, Yorktown Hts., N.Y.

10. Kent, W. Data and Reality, North Holland, 1979.

11. Smith, J.M. and Smith, D.C.P. "Database Abstractions: Aggregation and Generalization." ACM TODS, Vol. 2, No. 2, June 1977, pp. 105-133.

12. Chen P.P.S. "The Entity-Relationship Model - Toward a Unified View of the Data." ACM TODS, Vol. 1, No. 1, March 1976, pp. 9-36.

13. Codd, E.F. "Extending the Data Base Relational Model to Capture More Meaning." IBM Research, Yorktown Hts., N.Y., Report RJ2472.

14. Korzybski, A. Science and Sanity; An Introduction to Non-Aristotelian Systems and General Semantics, 4th de., International Non-Aristotelian Library Publishing Co., Lakeview, Conn., 1958.

15. Hall, P. et.al. "Relations and Entities." in Modelling in Data Base Management Systems, ed. G.M. Nijssen, North Holland, 1976.

16. Senko, M.E. "DIAM as a Detailed Example of the ANSI SPARC Architecture." in Modelling in Data Base Management Systems, ed. G.M. Nijssen, North Holland, 1976.

17. Codd, E.F. "Further Normalization of the Data Base Relational Method." in Data Base Systems, Courant Computer Science Symposia, Vol. 6, Prentice-Hall, 1972.

18. Bernstein, P.A. "Synthesizing Third Normal Form Relations From Functional Dependencies." ACM Trans. Database Systems, Vol. 1, No. 4, December 1976, pp. 277-298.

19. Codd, E.F. "A Relational Model of Data for Large Shared Data Banks." Comm. ACM, Vol. 13, No. 6, June 1970, pp. 377-387.

20. Lipski, W. "On Semantic Issues Connected with Incomplete Information Databases." ACM TODS Vol. 4, No. 3, September 1979, pp. 262-296.

21. Codd, E.F. "Relational Completeness of Data Base Sub-Languages." in Data Base Systems, Courant Computer Science Symposia, Vol. 6, Prentice Hall, 1972.

22. Bateson, G. Mind and Nature, A Necessary Unity, E.P. Dutton, New York, 1979.

VOLUME II

# THE DIAM THEORY

# OF ALBEBRAIC ACCESS GRAPHS

CONTENTS

## 1.0 Introduction

This paper presents a universal model for describing the logical structure of any database or network of databases, and a theory of navigation that allows the automatic computation of the information obtained by any database navigation in implementation-independent (in this case relational) form. The assumptions upon which the theory are founded are simple, natural and necessary, resulting in the mathematical foundations of the science of representation independence, representation dependence and the mapping from the former into the latter.

The representation-independent model used here is the relational model [1] because of its wide renown, its standarization, and the fact that the algebra so essential to the graph theory presented here is best known in that context. But there is no essential problem with using any of the other well known models developed for the purpose of representation-independent modeling of data instead, such as the entity set model [2].

The representation-dependent modeling method presented here makes use of di-graphs with no particular constraints as to how they should be connected except that they should be faithful to the database being modeled. The generality of di-graphs can hardly be argued, since all logical data structure models including the

data description languages of CODASYL and IMS are essentially that. What is new in this approach is the disciplined approach to schema and state which are used in the same sense as in relational theory. A relation called the control is specified at each node of the graph to determine its multiplicity given a relational state. A projection of the control called the value specifies what data element (instance of an attribute) is located at a node, if any. The instance specification of an edge is then shown to be expressible as an algebraic join of the controls on either side of the edge. These three assumptions determine the entire theory that follows. Like any simple, but basic theory, it is rich in consequences.

This paper is organized as follows:

Section 2 presents some algebraic notation and transformation rules.

Section 3 presents the basic definitions for algebraic access graphs. Section 4 illustrates the modeling of many common database structures using them.

Sections 5 through 8 develop the theory of first order navigation, and the algorithm for computing the information resolved by a navigation. Section 9 is on the complexity of the algebraic expressions required to perform this analysis.

The graphs developed by the basic theory are as fine-grained as possible, so as to be perfectly general. Their nodes contain only atomic data elements. In Section 10, it is shown how to create a more aggregated global graph that surpresses local detail.

Section 11 is concerned with the application of traditional graph theoretic tools to algebraic graphs.

Section 12 places the algebraic graph theory in the context of query compilation, and section 13 is a bried outline of the only known complete compilation algorithm.

It should be emphasized that the ideas presented here did not develop in a vaccum, and are not the inspiration of the present author alone. DIAM theory was pioneered by the late Michael E. Senko, who was the first database practitioner to believe that the possible ways of arranging representations of information in a space were not arbitrary or lawless, but followed definite principles that could be codified. The DIAM I string model (which the present contribution replaces) was the first attempt to present such a theory at the level of logical connectivity of implementations. It required three types of strings, but had no explanation of why there were exactly three types, except that it seemed to work. Schneider [3] then showed that these three strings could be associated with the relational operators restrict, project and

## 2.0 Relational Algebra

The reader should be familiar with the relational model and the basic definitions of relational algebra, specifically: relational schema, domain, attribute, key, projection, join, equijoin, natural join, restriction (sometimes called selection), union, intersection and difference. The concepts of schema and state are important throughout this paper. A relational schema is a specification of a collection of relations together with the domains and role names of all attributes, keys of relations, and any integrety constraints. A relational state is a set of tuples for each relation of the schema, such that they obey all the conditions of the schema. All the pertinent definitions for relational algebra can be found in [1].

We have found it useful to introduce some new notation because we have pursued the manipulation of relational algebra more extensively than has previously been done in the literature.

(1) E is the unit relation which is the set containing the 0-tuple. Its cardinality is 1, unlike the empty set whose cardinality is 0.

(2) * is the cross product. For any relation R, E*R=R.

(3) !R is the list of all attributes of R. This notation can be used in specifing joins, restrictions and projections.

(4) <R;A> is the projection of R onto the attribute list A. (Codd's notation is R[A].)

(5) (R;rc) is a restriction, where rc is a restriction condition, i.e. a list of equality or inequality expressions comparing attributes of R with constants or other attributes. (Codd's notation is R[rc].)

(6) R[jc]S is the join of R and S where jc is a join condition, i.e. a list of equality or inequality expressions comparing attributes of R to attributes of S. (Codd's notation is the same.) The advantage of using three different bracket styles for these three operators is visual clarity. Also, we have delimited the projection and restriction precisely by the placement of brackets.

(7) R[i]S is the identity join of R and S. It is meaningful only when R and S are union compatible, and is defined as R[!R=!S]S.

(8) R:[jc]S is the restriction of R by joinability to S. Its definition is <R[jc]S;!R>.

(9) Any algebraic expression may be called a <u>generalized restriction</u> of R if it defines a relation which is always a subset of R (and therefore union-compatible with R.) For example, R:[jc]S and R-S are generalized restrictions of R.

(10) Any algebraic expression may be called a generalized join of R and S if it defines an expression which is always a subset of (and therefore union-compatible with) the cross-product R*S. $\langle R[jc_1]S[jc_2]T;!R!T\rangle$ is a generalized join of R and T. Given any generalized join of R and S, there is always an expression T, such that the given generalized join can be written as $\langle R[jc_1]T[jc_2]S;!R!S\rangle$ where $jc_1$ and $jc_2$ are equi-join conditions. The expression $[jc_1]T[jc_2]$ is called a compound equi-join condition. For example, given the join $R[A\underline{<}B]S$, its compound equi-join condition is $[A=A]\ R[A\underline{<}B]S[B=B]$. If we want a join condition that eliminates redundant attributes, we can speak of the corresponding compound natural join condition.

The following equivalence transformations are useful for the graph theoretical work that follows:

(1) $(R \cup S)-T \quad <=> \quad (R-T) \cup (S-T)$

(2) $R-(S \cup T) \quad <=> \quad (R-S)-T$

(3) $\langle R \cup S;A\rangle \quad => \quad \langle R;A\rangle \cup \langle S;A\rangle$

(4) $\langle R;A\rangle \cup \langle S;A\rangle \quad => \quad \langle R \cup S;A\rangle$ when R,S union compatible.

(5) $(R \cup S)[jc]T \quad <=> \quad (R[jc]T) \cup (S[jc]T)$

(6) $(R \cup S;rc) \quad <=> \quad (R;rc) \cup (S;rc)$

(7) $(R \cap S)$ <=> R:[i]S, therefore intersection is a derrived operation and will not be treated separately.

(8) (R-S)[jc]T <=> R[jc]T-S[jc]T

(9) (R-S;rc) <=> (R;rc)-(S;rc)
<=> (R;rc)-S

(10) <R;A>[jc]<S;B> => <R[jc]S;A,B>

(11) <R[jc]S;A,B> => <R;A>[jc]<S;B> when jc references only attributes of A and D.

(12) (<R;A>;rc) => <(R;rc);A>

(13) <(R;rc);A> => (<R;A>;rc) when rc references only attributes of A.

(14) $(R[jc]S;rc_1)$ <=> $(R;rc_2)[jc](S;rc_3)$ where $rc_1$ is the union of the restrictions of $rc_2$ and $rc_3$. (Either $rc_2$ or $rc_3$ could be empty.)

(15) <R[A=B]S;R/A> <=> <R[A=B]S;S/B>

Rules (1) through (14) are distribution rules which can put an expression into a simpler form as has been noted by Smith and Chang [4] and others. Rule (15) is an important equivalence for algebraic graph theory because it shows how factored values are realized in a database. (Examples of this follow in Section 4.)

There are no general distribution rules for moving a projection either above or below a difference. Rules 1,2,3,5 and 6 show that a union can always be placed above (outside) any other operator. Rules 8,9,10 and 12 show that joins and restrictions can always be placed below differences and projections. Rule 14 shows that join and restriction can always be interchanged.

Any formula of relational algebra can therefore be transformed using these rules into a form in which unions are the highest operators, differences and projections come next, then joins, and finally restrictions. (This is reading from the top to the bottom of the expression viewed as a tree and is inverse to the order of performing the operations.) Since differences and projections cannot be passed through each other in general, the alternation of difference and projection serves as a method of classifying formulas, which is analogous to the Kleene arithmetic hierarchy for first order logical formulas. We do not need to study the entire hierarchy, but it is worth mentioning the class of formulas that do not contain difference (the P class), and the class of formulas in which projections are always below differences (the DP class). The class P is closed under all algebraic operations except difference. The class DP is closed under all operations except projection, and in certain important cases is closed under projection.

As an example of a more complex formula, consider

T-T:[A=A](R-R:[B=B]S)

It can be normalized by placing all joins below projections and differences, which results in the equivalent formula

T-<(T[A=A]R)-<T[A=A]R[B=B]S;!T!R>;!T>.

This is a DPDP formula and has no simpler form. It is our present opinion that the efficient compilation of queries more complex than DP is reasonably left to artificial intelligence specialists rather than database practitioners.

Let us call the algebra of restriction, join, projection, difference and union first order relational algebra. Then there are quite reasonable algebraic operators that are not first order. The most commonly found in realtional query languages are the aggregate operators COUNT and SUM which make possible queries such as: How many accounts are 90 days past due, and what is the total amount receivable from these accounts? (Aggregates are high order operators because they apply an operation to a class of individuals, not because they invoke arithmetic operations.) One can also consider ordinal operators: Who are the fifth through twelth most senior employees in department X?, and connectivity operators: Find all connecting flights from Eureka, California to Syracuse, New York that do not reverse direction. In this paper, we do not consider such languages for compilation, but leave that for future research.

## 3.0 The Algebraic Graph Model: Definition

The notion of schema and state is as important to the algebraic graph model as to the relational model. We start by defining an algebraic graph schema over a relational schema as consisting of the following:

(1) A labeled directed graph such that each edge and each node has a distinct label. (Unlike most definitions of a directed graph, we do not prohibit an edge from connecting a node to itself. Also, we do not prohibit more than one edge connecting a given pair of nodes in the same direction, but each such edge will have a distinct label.)

(2) A relation associated with each node of the graph called the control of the node. The control can be any relation defined by an algebraic expression over the relations of the given relational schema. (In this paper we consider only relations that are definable using first order algebra, but one could do otherwise.)

(3) For certain nodes of the schema called value nodes, a value which is a single attribute projection of the control of the node.

(4) A natural join condition (simple or compound) associated with every edge of the graph. This must produce a meaningful algebraic expression when flanked on the left and right with the controls of the origin node and destination node of the edge respectively.

A node whose control is E (the unit relation) is called an entry point of the graph.

This completes the definition of the algebraic graph schema, and we now proceed to define an algebraic graph state. Given a relational schema, a relational state, and an algebraic graph schema, an algebraic graph state can be constructed as follows:

(1) For each schema node, there will be a set of instance nodes, one for each tuple in the instance table of the control of the schema node.

(2) If the node in schema is a value node, then each instance node will have an actual value which is the specified projection of its controlling tuple.

(3) For each edge in schema, there will be a set of edges in instance. An instance of the origin node will be linked to an instance of the destination node whenever their respective tuples satisfy the join condition of the edge.

The preceeding definitions are the foundation of algebraic graph theory, and it is essential that their significance be understood. From the experience of oral presentations, this author is aware that the reader will probably find them strange and unnatural, and that no amount of philosophising will alter this. The author wished to apologize for not knowing the proper pedegogical means of

explaining the theory. Fortunately, everyone who has taken the trouble to study the examples in Section 4 has in retrospect found these definitions to be simple and obvious.

A corollary of these definitions is that a relational schema, a graph schema over the relational schema, and a relational state uniquely determine a graph state. (Figure 7 is a detailed example of this corollary).

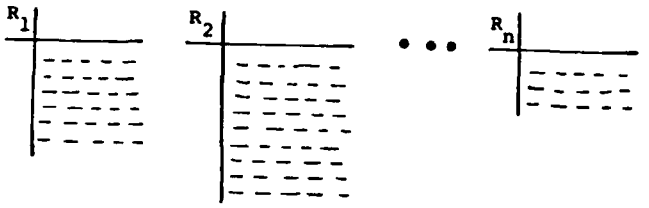
The following conventions are used for drawing schemas: Each node is depicted as a small circle with the node label inside the circle. The control of the node will always be written above or to the left of the node. The value, if there is one, will be written below or to the right of the node. Each edge is depicted as a line connecting two nodes and having a barb to indicate its direction. The edge label is written above or to the left of the edge, and the join condition is written below or to the right of the edge.

A relational schema:

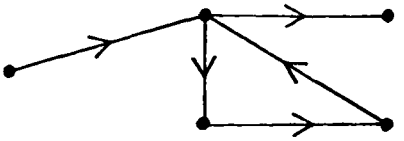
$R_1(\ldots\ )$
$R_2(\ldots\ )$

$R_n(\ldots\ )$

a relational state:




and a graph schema:



determine a graph state.

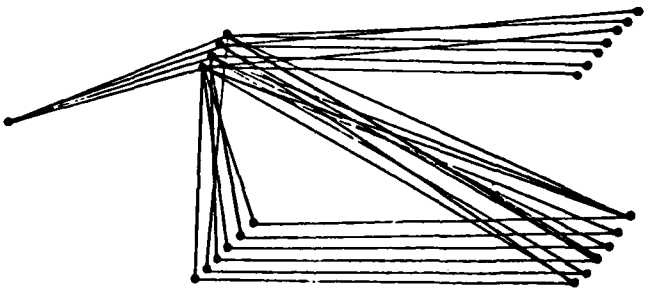


Figure 1

## 4.0 Modeling Data Structures

While the formal properties of a model may be treated mathematically, the application of the model depends on practical experience. The intended use of algebraic graphs is to model data structures at a level of abstraction which is neither implementation-independent nor a detailed specification of record layouts on a magnetic surface medium, but is somewhere between these two. Algebraic graphs are useful because they are simple of definition, because they appear to be universal for modeling data base applications (which is what we shall illustrate in this chapter), and because they divide the process of compiling database transactions into two distinct phases, the first of which is global and topological, and the second of which is local and concerns access methods and encodings.

The first four examples described here model different implementations for a database consisting of the single relation R(A,B,C) where A is the key. Figure 2 could represent a file consisting of records each of which contains one tuple of relation R. The file could be ordered sequentially on the key, A, but this is not indicated. The file could be packed sequentially, or the records could be stored according to some hashing scheme, or there could be a threaded list of records or some combination thereof. There

is also the possibility that the tuples of R are not stored as records, but are list structures with pointers from A to B and from B to C. Figure 1 only specifies that all the tuples of R are accessed from a single entry point, and that there is some method of traversing the attributes A, B and C of each tuple in that order.

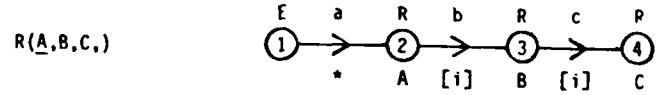



Figure 2

Figure 3 illustrates the same file with the addition of a secondary index on attribute B. The graph shows that from entry point 5, one may get to node 6 whose control is <R;B>, i.e. a node whose cardinality in instance is the cardinality of the attribute B which is presumably smaller than the cardinality of R since B is a non-key attribute. From any instance of node 6, one may access all R-tuples having that B value, therefore the join condition of the path is [B=B]. A useful implementation would most likely order the edge d according to attribute B, and edge e according to attribute A.

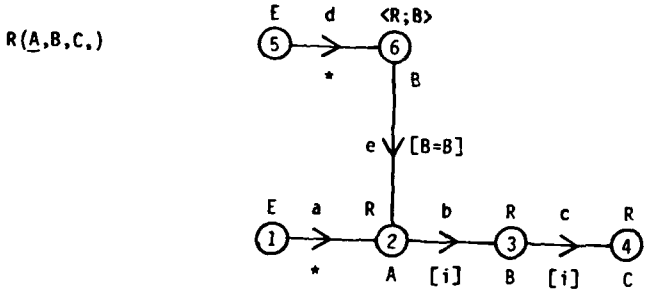



Figure 3

Figure 4 illustrates the same relation implemented so as to store the values of attribute C non-redundantly. This might be useful if each value of C was very bulky, and the cardinality of <R;C> was much smaller than the cardinality of R. In this type of storage, the edge c cannot be implemented by continguity.

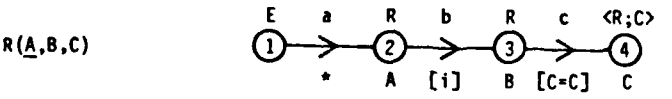



Figure 4

Figure 5 again represents the same relation R, stored as three separate files which might be geographically remote from each other. The relation R is partitioned by restriction distribution, and one portion of it is further fractioned by projection distribution.

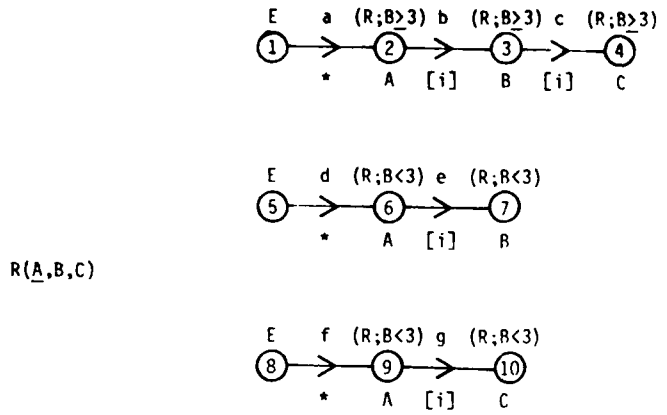



Figure 5

Figure 6 is the relation S(A,B,C) which has a binary key. It accesses groups of tuples having a common A value from the entry point, then accesses the individual tuples having that A value but different B values from each of those points. Very likely an implementation would sort on A, then on B.

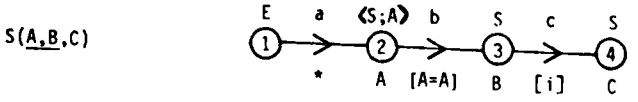
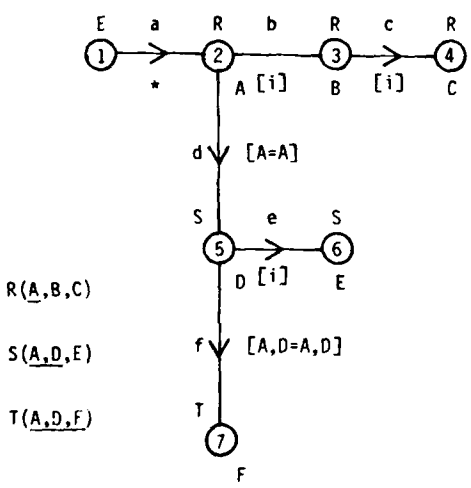



Figure 6

Figure 7 is an implementation of a hierarchical storage of three relations by means of join collocation. This is a complete storage schema for the relational schema shown only if the relational schema is constrained by S=S:[A=A]R and T=T:[A,D=A,D,]S otherwise there is no place to store certain tuples. Figure 7 also shows a relational state and a corresponding graph state for this schema.




| R | A | B | C |
|---|---|---|---|
| 1 | X | U |
| 2 | Y | V |
| 3 | Z | W |

| S | A | D | E |
|---|---|---|---|
| 1 | 3 | X |
| 1 | 4 | X |
| 2 | 3 | Y |
| 2 | 5 | Z |

| T | A | D | F |
|---|---|---|---|
| 1 | 3 | M |
| 1 | 3 | N |
| 1 | 4 | L |
| 2 | 5 | M |
| 2 | 5 | L |

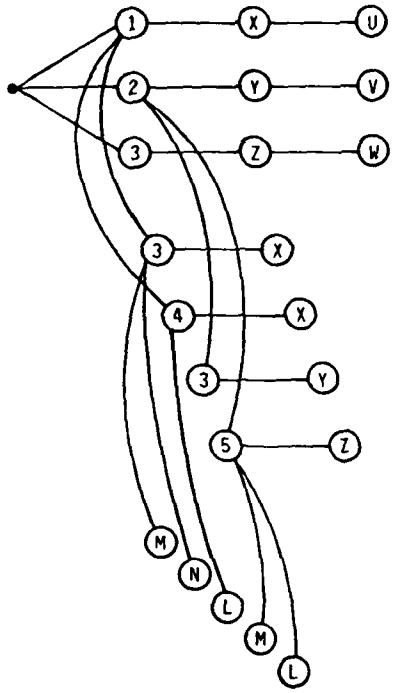



Figure 7

Figure 8 shows another hierarchical storage schema, with the distinction that the key of relation S is distinct from that of R rather than being pyramided. Since each S-tuple may be joined to several Rtuples, these may be stored redundantly. Those S-tuples not joinable to R-tuples are stored separately.

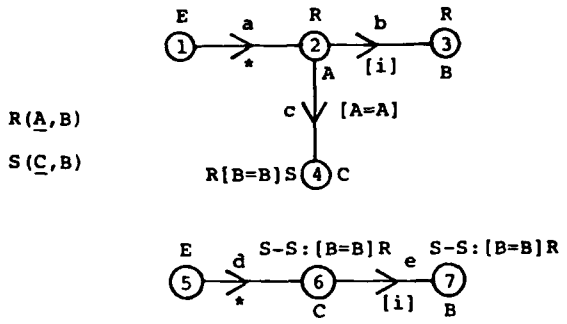



Figure 8

Figure 9 shows a file with a secondary index on attribute B which is multi-valued dependent on A. This makes for a compound join condition on edge e, because B is not an attribute of R. A compound join condition may invoke information that is not present at either end of the edge, in this case, the relation S.

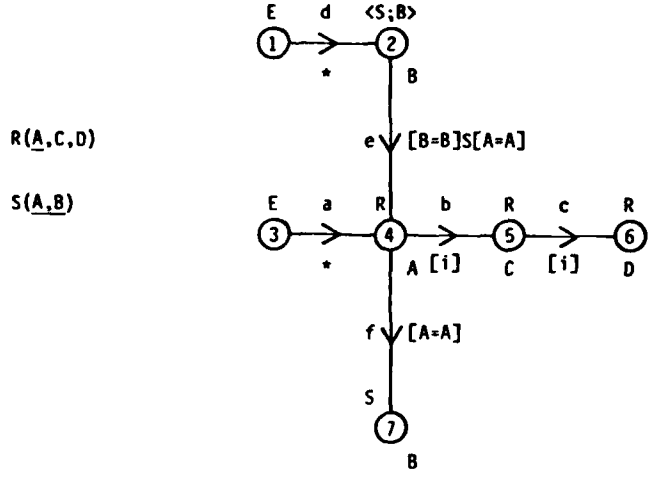



Figure 9

Figure 10 is a storage scheme for a single relation R, having a secondary index on the attribute B, but the index is partial, and only for those tuples where C=3.

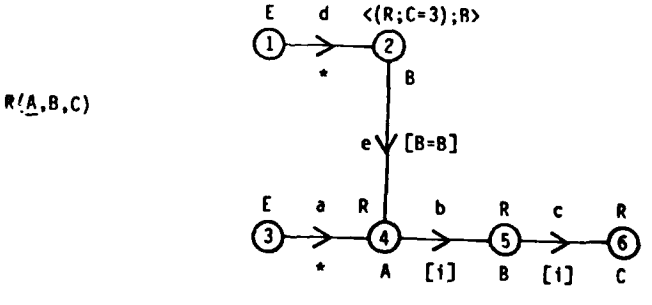



Figure 10

Figure 11 is a network storage scheme for three relations. In CODASYL terminology, there are two owner coupled sets: in one, S is the owner and SP is the member; in the other, P is the owner and SP the member. SP is an intersection set which does not contain either of the key attributes, S# and P#, of the SP relation because these are factored onto the two owner records respectively. This schema also illustrates the possibility of information schemas with cycles.

S(S#, SNAME, STATUS, CITY)

SP(S#,P#, QTY)

P(P#,PNAME, COLOR, CITY)

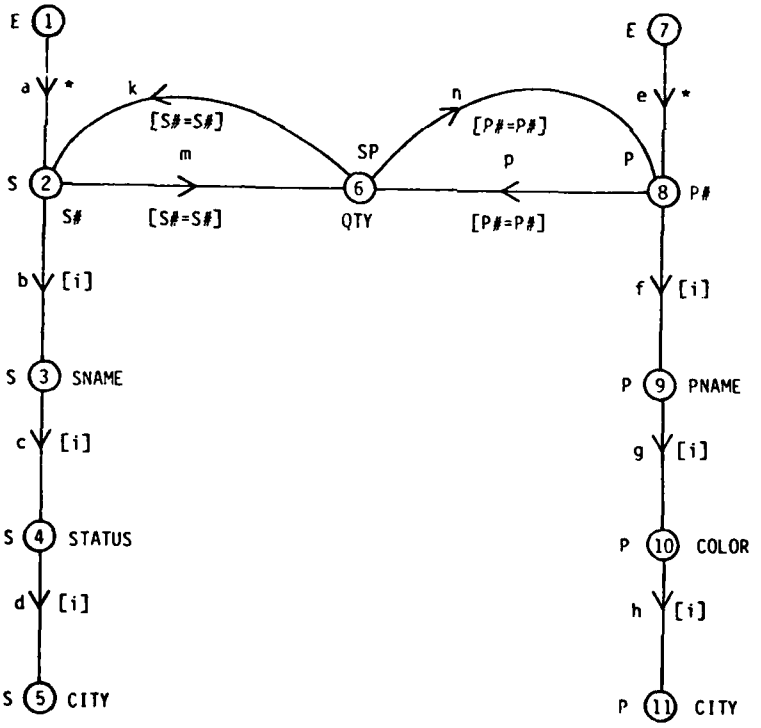



Figure 11

The examples of this section have been selected to illustrate the comprehensive modeling power of algebraic graphs, and their ability to corelate relational and network or hierarchical views of the same data.

## 5.0 Linear and Tree Navigation

The next few chapters describe the navigations of an algebraic graph. The purpose is to characterize navigations, and develop algorithms that compute the information extracted by a navigation. For the present, we shall describe a navigation as if it were a set operation, i.e. one in which a group of instance paths were being explored simultaneously.

To convert such a navigation into a more realistic database navigation, one must specify the ordering of those edges which represent one-to-many traversals and then serialize the navigation. Furthermore, one must plan the storage of the resolved information in temporary files and actualize the processing of these as sequences of sorts and merges when necessary. It is not our intention to minimize the importance of these issues which have been studied by Smith and Chang [4], Rothnie et. al. [5] and Yao [6]. It is simply a fact that all navigations can be enumerated by first enumerating all of them at the graph level, and then further refining each of those at the level of sequential operations.

The set of navigations to be described here we shall call first order navigation. It appears to be rich enough to allow general DP query compilation, but not rich enough to permit the sort of heuristic searches that might be created by a compiler of highly quantified queries acting with specific knowledge of the application.

A navigation is a set of paths some of which are dependent upon others and must therefore be performed after those they are dependent on. A path in turn is made of steps, which are numbered sequentially. (Our use of the work path here corresponds with the graph theoretic meaning, with some details added).

An unrestricted entry path is the simplest type of path traversal. It is a path starting from an entry node. It is permitted to contain loops and cycles.[1] It is unrestricted in the sense that all the instances corresponding to the schema path are to be traversed. Every path has a control and a value associated with it. For an unrestricted entry path that traverses edges $e_1$ through $e_n$, let $[j_1]$ through $[j_n]$ be the join conditions of these edges respectively, and let $C_o$ through $C_n$ be the controls of the nodes encountered in that order. Then the control of the path is the expression $C_o[j_1]C_1 \ldots [j_n]C_n$. To avoid generating large expressions with redundant attributes, we shall use natural joins instead of equi-joins. This causes R[i]R to simplify to R for example. The value of the path is the projection of the control onto those attributes which are encountered as values along the path.

[1] Loops and cycles in a schema do not necessarily correspond to loops and cycles in instance, because different instance nodes may be encountered when the path encounters a previously traversed node. (See Figure 22).

Example: Over the algebraic graph of figure 2, consider the unrestricted entry path a,b. The control is E*R[i]R which using the natural join simplifies to R. The values A and B are encountered on this path, therefore the value is <R;A,B>.

Example: Over figure 11 consider the unrestricted entry path a,m,n. The simplified control is S[S#=S#]SP[P#=P#]P. The value of the path is <S[S#=S#]SP[P#=P#]P;S/S#,SP/QTY,P/P#>. Here, we have had to qualify attributes by the relation that they came from using a slash. But since S/S# and SP/S# are equivalent when joined (Section 2, rule 15), we could have written SP/S# for S/S#. Similarly, P/P# can be replaced by SP/P#. If the relation SP is hierarchically subordinate to S and P, i.e. the relational schema has the constraints SP=SP:[S#=S#]S and SP=SP:[P#=P#]P, then the value of this path simplifies to SP.

An entry path does not have to correspond to all instances of the path in schema, but may be a restricted entry path. Restrictions based on a particular attribute may occur only at a node where that attribute appears as value. For example, using figure 11, we may traverse path a, then restrict by S#=237 because S# is the value of node 2, then take the edges m and n. This path can be denoted as a,(R;S#=237),m,n. The control of a restricted path is computed in the same manner as the control for an unrestricted path

except that where a restriction occurs at a node, the restricted rather than the full control of that node should be used. In this case, the path control is (S;S#=237)[S#=S#]SP[P#=P#]P. The value expression is obtained from the control expression as before, and under the hierarchical subordination constraints mentioned in the above, the value is (SP;S#=237).

When a path is restricted at several nodes, initial segments of it may contain information, than is not indicated by the value of the entire path. For example, over figure 11, the path a,(S;S#=237),m,n,(P;P#=16) has the value (SP;S#=237,P#=16), whereas the initial segment a,(R;S#=237),m has the value <(SP;S#=237); S#,QTY)>.

Restrictions are not limited to comparisons of the value of a node with a constant, but may occur through comparison of the value of a node to values in a set of previously resolved information. For example, suppose we wish to resolve the query (R;B=2) over the graph of figure 5. One way to do this is to execute the path d,e,(R;B=2) which resolves the intermediate result $T_1$=<(R;B=2);A,B>. Then one can execute the path f,R:[A=A]$T_1$,g. The value of this path is $T_2$=<R:[A=A]$T_1$,A,C>=<(R;B=2);A,C>. The answer to the query is $T_1$[A=A]$T_2$.[2]

[2] For a discussion of an application of this "feedback" See [6], p. 139.

In addition to paths that start at entry points, there are paths that start at a node which has been reached by a path previously traversed. This we shall call grafting. The point of origin of a grafted path is generally a restriction of the instances of this node rather than being all of them. Therefore, it is important to specify the point of origin not just as a particular node of the graph, but as a particular step of a previous path. For this reason, we now develop a more detailed format for specifying paths.

Each path consists of a distinct label and a set of sequentially numbered steps. Steps of type ENTER, EDGE, RESTRICT and GRAFT will be discussed here, and INTERSECT and DIFFERENCE will be introduced later. Every step has a location which will be a node, and some steps have antecedents which will be steps of previous paths, or in the case of a restrict, values of previous paths. The complete repertoire is as follows:

| | |
|---|---|
| ENTER entry | ; location is entry. |
| EDGE edge TO node | ; location is node, and edge must traverse from the location of the previous step to node. |
| RESTRICT expr | ; location is the same as the previous step, and expr is a generalized restriction on the control of the location of the step, and which depends only upon (i) the value attribute of that location, and |

value attributes of previous nodes of the same path, (ii) constants, and (iii) previously resolved relations.

An EDGE followed by a RESTRICT may occur as a single step. This means that only those instances determined by the restrict were arrived at by the edge traversal.

GRAFT path,step ; location is the same as that of the antecedent step.

INTERSECT path1,step1,path2,step2

; both antecedents must have the same location, which is also the location of the INTERSECTION.

DIFFERENCE path1,step1,path2,step2

; both antecedents must have the same location, which is also the location of the difference.

The first step of any path is always ENTER, GRAFT, INTERSECT or DIFFERENCE. The remaining steps are always EDGE or RESTRICT or EDGE-RESTRICT.

For the purpose of analysis, we shall consider grafted paths to be entry paths by the device of tracing their lineage back to an entry point. This permits us to compute the control and value of a grafted path in the same manner as entry paths.

**Example: Over figure 11, resolve the query <(S;S#=3)[S#=S#]SP; SNAME,QTY,P#>. (Get the name and all shipments for supplier S3). This query can be resolved as follows: (figure 12):**

```
PATH 1:     1:  ENTER 1
            2:  EDGE a TO 2
                RESTRICT (S#=3)      ; This is a combined step.
            3:  EDGE b TO 3
            CONTROL:  (S;S#=3)
            VALUE:  T1=<(S;S#=3);S#,SNAME>

PATH 2:     1:  GRAFT PATH 1, STEP 2; CONTROL IS (S;S#=3)
            2:  EDGE m TO 6
            3:  EDGE n TO 8
            CONTROL:  (S;S#=3)[S#=S#]SP[P#=P#]P
            VALUE:  T2=(SP;S#=3)

QUERY:  <T1[S#=S#]T2;SNAME,P#,QTY>=<(S;S#=3)[S#=S#]SP;SNAME,P#,QTY>
```

Figure 12

When paths are grafted onto other paths, there is the possibility that more information is discovered by a navigation than is revealed by the value expressions of the individual paths. The graph of Figure 13 shows a tree which forks with neither branch having a key among its values. The path a,b has a control which is R, and a value which is <R;A,B>. The grafted path c (equivalent to a,c) has a control R, and a value <R;A,C>. These two projections cannot be joined externally to recover the relation R because they do not contain a common key. However, a navigation which keeps track of which instances of the left path correspond to which instances of the right path does recover the relation R.

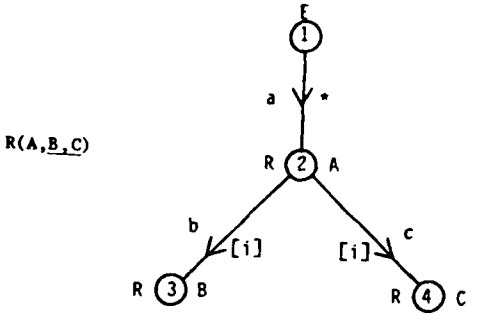



Figure 13

When two paths share a common node (which is always the case when one of them is grafted from a step in the other one) there is a common control and a common value which may contain information missing from the separated value expressions. The common control is the join of the separate controls, using as join condition, all the attributes of the control of the common node (and not just the value of the common node.) In our previous example, both separate controls are R, and the join is R[!R=!R]R=R. The common value is the common control projected onto the values of both paths. In our previous example, the common value is R because all three attributes are now available on the resulting tree. This process may be continued, leading to common expressions for trees of any complexity.

A path, tree, or subtree can be said to be self-indentifiable if its value contains at least a key of its control. Grafting of two paths and/or trees leads to a common value containing more information than the separate values if and only if at least one of the two structures being joined is not self-identifiable. Generally, good database design requires that structures be easily identifiable, but it is not necessarily the case that every record is self-identifiable. Figure 11 shows a CODASYL implementation in which the key of the SP relation which is necessary to identify an SP record can be found only by means of owner coupled pointers.

In this chapter, we have shown how to derive control and value expressions for any navigation which is a tree structure emanating from an entry point. The control gives an algebraic description for each path or tree instance, and determines the cardinality of that structure. The value specifies the information learned by executing the navigation.

## 6.0 Retraction Paths

The traversal of a database by an access engine is not accurately modeled by paths over a digraph because programs are capable of remembering a previous point of a traversal and backtracking. To model this, we define the notion of a retraction path as follows: If a,b,c etc. are edges, then $a^-,b^-,c^-$ etc. are these same edges traversed in the backward direction. A retraction path is a sequence of edges and inverse edges such that their incident nodes align properly, and such that it can be reduced to a path without inverses by a series of cancellations of pairs e, $e^-$ where e is an edge and $e^-$ is its inverse.

Examples of valid retraction paths over figure 11 are:

$$a,b,c,c^-,b^-,m,n,n^-,k,b$$

$$a,m,n,p,p^-,f,f^-,n^-,k$$

Invalid examples are:

$$a,m,n,p,m^-,b$$

$$a,k^-,k,b$$

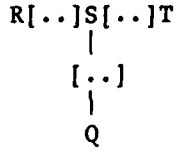
A consequence of the definition is that any initial segment of a retraction path is a retraction path, but this is not generally true of terminal segments.

When a retraction path starts at a graft point rather than an entry point, the retraction may involve portions of the antecedent

path and earlier paths as well. The analysis of a path as a valid retraction path can be understood only when it is traced back to an entry point.

The control of a retraction path is not found by creating a linear join sequence for reasons that will be stated presently. Rather, a T-shaped join must be formed. Inverse edge tranversals do not contribute to this join. Therefore, after a sequence of these has occured, the next positive traversal must be joined not to the end of the expression formed thus far, but to the node from which the retracted stub departed. An algorithmic statement of this rule is messy, but the examples should make it clear.

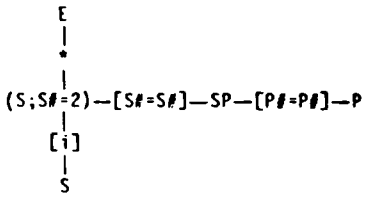
This looks simple, and is not difficult to understand, but gives rise to a notational problem. Until now, we have had to write only linear strings of joins. But in this case, Q is joined to somewhere in the middle of what may be a long formula. This leads us to want to write formulas that are no longer linear, e.g.:




One could write $(R[jc_1]S[jc_2]T)[jc_3]Q$, provided that $jc_3$ could reference attributes of S conveniently. We have yet to find a notation that works in all cases and is readable by people as well.

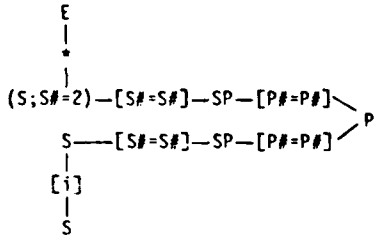
Inserting a branching join [jc]R into a join sequence is quite different in computational effect than inserting the linear sequence [jc]R[jc]. An example should make this clear. Over figure 11, consider separately the two navigations (i) a,(S#=2),m,n,$n^-$,$m^-$,b, and (ii) a,(S#=2),m,n,p,k,b. The first path, which uses retraction, has the control (S;S#=2)[S#=S#]SP [P#=P#]P, and the value <(S;S#=2) [S#=S#]SP;S#,QTY,P#,SNAME>. The second path has the control (S;S#=2)[S#=S#]SP[P#=P#]P[P#=P#]SP[S#=S#]S, and the value which is the projection of this onto $S\#_1$,$QTY_1$,P#,$QTY_2$,$S\#_2$,SNAME (See Figure 14). The first path resolves the query: Get the name of supplier 2, and the quantity and part number of all his shipments. The second path resolves the query: Get the quantity and part number of all shipments for supplier 2, and then for each part supplied by supplier 2 get all shipments of that part by any supplier, specifying quantity, supplier number, and supplier name.

The control of a, (S#=2),m,n,n⁻,m⁻,b is:




which simplifies to (S;S#=2)[S#=S#]SP[P#=P#]P.

The control of a,(S#=2),m,n,p,k,b is:




which simplifies to
(S,S#=2)[S#=S#]SP[P#=P#]P[P#=P#]SP[S#=S#]S.

Figure 14

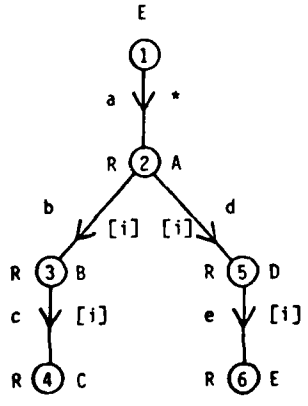
In this example, n,p is quite different than n,n⁻, although n and p have the same join condition. The important point is that a retraction only follows those join instances back from which the access engine came, and not all possible edges from which it could have come.

It is obvious that any tree can be traced by a single retraction path. But whether one wants to do this or to use grafting will depend on whether the various restrictions are to be considered in one long sequence or whether some of them are intended to apply to only portions of the tree. Any tree navigation described by a single path will have a single value expression descriptive of the entire tree. But when different restrictions apply to different portions of a t·ee, then different portions will have different information descriptions.

Figure 15 is an artifical example of the complexity of combinations of grafting and retraction.

R(A,B,C,D,E)

Key is C,E




```
PATH 1      1: ENTER 1
            2: EDGE a To 2
            3: EDGE b To 3
            4: RESTRICT (B=3)
            5: EDGE c TO 4
            CONTROL:  (R;B=3)
            VALUE:    <(R;B=3);A,B,C>

PATH 2      1: GRAFT PATH 1, STEP 4; CONTROL IS (R;B=3)
            2: RETRACT b- To 2
            3: EDGE d To 5
            4: RESTRICT (D=4)
            5: EDGE e To 6
            CONTROL:  (R;B=3,D=4)
            VALUE:    <(R;B=3,D=4);A,B,D,E>

COMMON CONTROL:  (R;B=3,D=4)
COMMON VALUE:    (P;B=3,D=4)
```

Figure 15

## 7.0 Intersection of Paths

There are certain graphs that cannot even in principle be navigated using only tree searches, and others which can be navigated more efficiently by other methods. Figure 16 is an example of the former. This graph has no trees, the two posssible entry paths resolve the values <R;A,C,D> and <R;B,C,D> respectively. But these cannot be joined externally to yield R because there is no common key.

An implementation based on this graph can be navigated by means of intersection on the address of node 3. From the path a,b,e one constructs the relation X(A,C,address of node 3,D), and from the path c,d, the relation Y(B,C,address of node 3). These can be joined by an external sort and merge to form the relation R. This is possible because the control of node 3 is ∀, and therefore the address of node 3 acts as an internal (nonsemantic) key for R.

When two paths (or more complex structures) intersect in this manner, and the intersection is performed externally, a <u>common control</u> is obtained by joining the individual path controls using all the attributes of the control of the intersection node. The <u>common value</u> is the projection of this onto the value attributes of both structures. The common control of a,b,e and c,d of figure 16 is R, and the common value is R.

This situation is mathematically equivalent to the common control and value of grafted paths, but in this case, the common control and value are not actualized unless an external address intersection is performed. Therefore, in our notation for navigations, we will allow INTERSECTION as the first step of a path. Such a path has two antecedents each of which is a step of a previous path, both being at the same location (node). The starting control for such a path is the intersection of the two step controls of the antecedents.

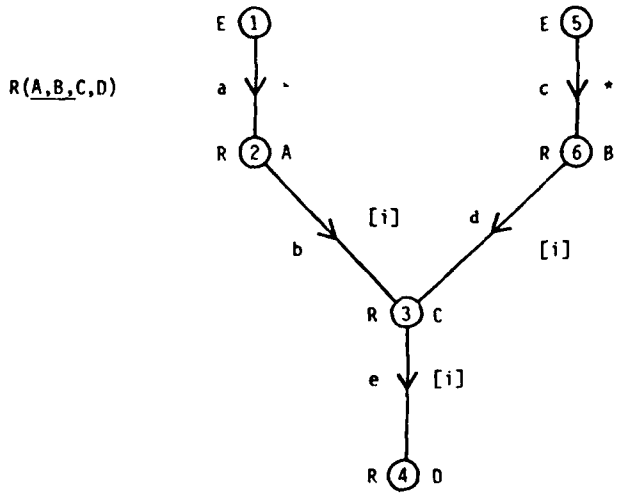



Figure 16

The combination of grafts, retractions and intersections can give rise to topologically complicated structures that are difficult to visualize. One must be careful to distinguish a genuine intersection or graft point from the situation of two paths crossing but not affecting each other. No additional mathematical tools are needed to analyze such situations. Each individual path has its control and value, and each collection of paths having connectivity has a common control and value based on the joins of its graft and intersection points.

Database modeling and navigation does not usually create path complexes of enormous complexity.

## 8.0 Differences of Paths

The use of linear paths, retracting, grafting and intersecting is a complete set of operations for navigating a database, but it is not the most efficient set. The collection of techniques and heuristics that could be used to optimize searches under special conditions is open-ended. The use of set differences is the only additional technique that we have considered as a tool for a generalized database access compiler.

The instance description of any step of a path is called a step control. It is obtained by taking the control of the initial segment of the path up to and including that step (so as not to include the effect of restrictions occuring after that step), and projecting this down to the attributes of the control of the location of the step. This always results in a generalized restriction of the node control, but mathematically may be complicated.

When two different step controls occuring at the same node have been created by navigation, then their set difference forms another step control which may be used as the origin of a new path. Such a path, called a difference path, will have two antecedents, a positive and a negative one. In tracing the lineage of a difference path to its origin, only the positive antecedent should be taken.

The control of a difference path can be computed by considering its lineage, and using the negative antecedent as a single restrictive condition at the point where the DIFFERENCE operation occurs. Let the node controls of the lineage, taken with all indicated restrictions be $C_o$ through $C_n$ where the DIFFERENCE occurs at $C_k$, and the path joins are $[jc_1]$ through $[jc_n]$. Let the step control of the negative antecedent step be D. Then the control of the path is $C_0[jc_1]\ldots[jc_k](C_k\text{-}D)\ldots[jc_n]C_n$.

Figure 17 shows a navigation of figure 16 using a difference path. Figure 18 shows why the difference path is a continuation of its positive antecedent, but does not have any lineage derrived from its negative antecedent.

```
PATH1:    1: ENTER 1
          2: EDGE a TO 2
          3: RESTRICT (A=3)
          4: EDGE b TO 3
          CONTROL:  (R;A=3)
          VALUE:    <(R;A=3);A,C>

PATH2:    1: ENTER 5
          2: EDGE c TO 6
          3: EDGE d to 3
          CONTROL:  R
          VALUE:    <R;B,C>

PATH3:    1: DIFFERENCE PATH2,3 - PATH1,4
          2: EDGE c TO 6
          CONTROL:  (R;A≠3)
          VALUE:    <(R;A≠3);B,C,D>
```

Figure 17

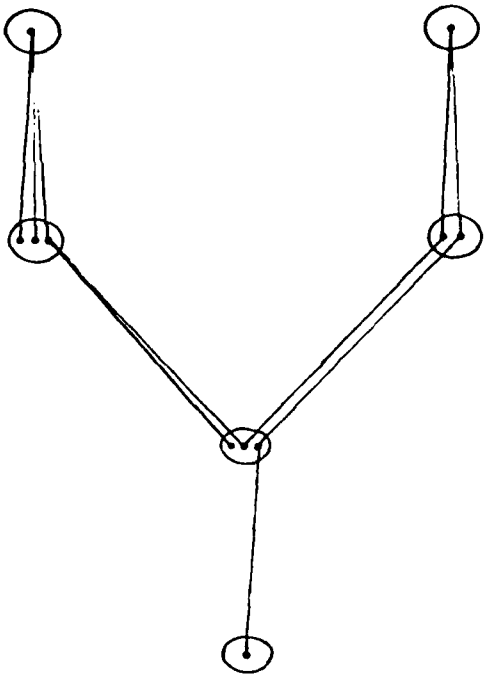


Figure 18

Difference is commonly used in combination with retraction. For example over figure 11, consider the query: Get the names of all parts for which there are no shipments of over 100 from any one supplier. This is resolved by Figure 19.

```
PATH1:    1: ENTER 7
          2: EDGE e TO 8                    ; control is P
          3: EDGE p TO 6
          4: RESTRICT (QTY<100)
          5: RETRACTION p- TO 8             ; control is P:[P#=P#](P;QTY>100)


PATH2:    1: DIFFERENCE PATH1,2 - PATH1,5
          2: EDGE f to 9
          CONTROL: P-P:[P#=P#](SP;QTY>100)
          VALUE:   <P-P:[P#=P#](SP;QTY>100);P#,PNAME>
```

Figure 19

## 9.0 Algebraic Complexity of Navigation

Consideration of the complexity of algebraic expressions that arise as a result of navigation is important because expressions beyond DP are difficult to work with. In characterizing the complexity of a situation we must consider (i) the complexity of the algebra used to describe the graph, (ii) the complexity of the queries, and (iii) the types of navigation invoked.

(i) Our experience is that DP is a suitable class of algebraic expressions for describing existing databases. The reason that P is inadequate is that the actual need to represent LP or outer joins gives rise to storage structures that contain the residue of a given type of entity not joinable to some other entity (i.e. the orphans). (See Figure 8). It is certainly possible to distribute data based upon some prediction involving complicated quantification of its attributes, which would create restriction expresssions beyond DP, but we have not found this to be practiced.

(ii) We have targeted DP queries as a useful and feasible class to be compiled. Given such a compiler, one can answer higher order queries with less than optimal efficiency by resolving their DP sub-expressions and combining thesee with sort, merge, filter and coalesce techniques. To our knowledge, no one has proposed to do better than this even in a homogeneous environment.

(iii) The set of navigation techniques described here with the exception of DIFFERENCE should be called positive navigation. The controls generated by positive navigation combine the algebraic expressions found on the graph using only restriction and join. Therefore, positive navigation cannot result in controls that exceed the complexity of the graph. (The value may require one projection on top of this).

The addition of DIFFERENCE to the set of techniques results in full first order navigation which can result in controls of any complexity even over a P graph. This is because each DIFFERENCE operation requires both a difference and a projection (to get the inhibiting node control), and these can easily be pyramided. Therefore, we have restricted the use of DIFFERENCE to those situations where its need is directly derrivable from the query to be resolved, and will not lead to expression that exceed the complexity of DP.

## 10.0 Zone Analysis and Global Graphs

The graphs we have been using thus far have the smallest granularity, and therefore are the most universal, because we have permitted at most a single attribute or value at each node. Sometimes one would prefer a more global graph that surpressed certain local details. For example, CODASYL DDL makes use of a graph in which the nodes are entire records.[3]

The approach used here does not replicate this effect exactly, firstly because records are a concept that cannot be defined at the level of logical connectivity, being dependent upon contiguity of storage[4], and secondly, we require that zones as defined here be in first normal form, which means that a record with multi-valued elements will be represented by a hierarchy of zones. The desired effect is to have each zone have a control and a value, and to surpress internal connectivity within the zone with the assurance that it is both sufficient and free from surprises. To accomplish this, some preliminary definitions are necessary.

---

3 Algebraic graphs are more specific than CODASYL DDL because even if the nodes were made to correspond exactly, the algebraic graph theory precisely defines the criteria for the edge instances (or "set" inclusion), whereas this information may be embedded in applications in CODASYL.

4 The DIAM concept equivalent to a record is called a contiguous data group and is definable at the encoding level. (See [3]).

Definition: Consider edges of the following three kinds only: (i) selective: <R;A>[A=A]R, (ii) isomorphic: R[i]R, and (iii) projective: R[A=A]<R;A>. (The traversal direction is left to right). A path comprised only of these three types of edges is convex if no projective edge preceeds any selective edge. A graph or subgraph containing only these three types of edges is convex if all paths in it are convex. A retraction path is convex if it meets the same criteria as does a convex path when considered as a linear sequence, bearing in mind that the inverse of a selective edge is projective and visa versa.

A zone analysis of an algebraic graph consists of (i) a partitioning of the set of nodes of the graph into subsets called zones, and (ii) a partitioning of the edges of the graph into two classes, class A and class B such that the criteria (iii) through (viii) to follow are satisfied. Every node which is an entry point of the database or the destination node of a class A edge is called an entry point of its zone. Every node which is an origin of a class A edge is called an exit point of its zone. (iii) Every edge which is inter-zonal is of class A. (iv) Every edge in class B is selective, isomorphic or projective. (v) The nodes and class B edges of each zone are a convex subgraph. (vi) Every zone has a control and a value, such that each node control within the zone

is either the zone control or some projection of it, and the zone value is the set of all node values within the zone. (vii) For every zone, given an entry point and an exit point of the zone, there exists a tranversal starting at the entry point, visiting at least one node having each of the specified attribute values, and ending at the exit point, and this traversal is a convex path or convex retraction path consisting of class B edges only.

The zone analysis of an algebraic graph is not unique (For example, there is always the trivial analysis in which each node is a zone, and all edges are class A). It may be that there is a unique maximal zone analysis, and in any case, we believe that automatic techniques for zone analysis will be feasible and useful.

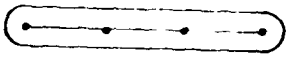
Given a zone analysis of an algebraic graph, the corresponsing global graph follows quite automatically, and consists of (i) nodes to represent each zone, (ii) zone controls and values at each node, (iii) all class A edges of the original graph together with their join criteria. (Note that there are some class A edges that were intra-zonal in the original graph, and these will appear as loops in the global graph). (iv) An indication for those zones having an external (database) entry.

A zone is self-identifiable if its value includes a key of its control. Zones which are not self-identifiable occur in hierachies

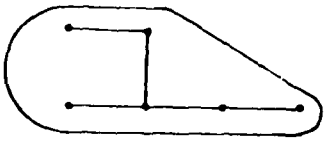
and networks, such that access paths through other zones supply the key.

Figure 20 shows zone analyses for several graphs taken from previous examples. The reader should be able to find controls and values for these zones, and determine those that are not self-identifiable. Figure 21 derives global access paths for some of these. The existence of an external (database) entry to a zone is indicated by the baton attached to the circle. Figure 22 shows two common situations in which an intra-zonal edge must be classified as type A.

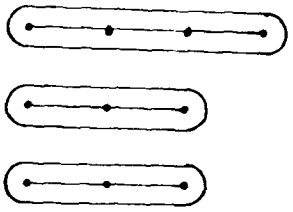
A global graph can be navigated in almost the same manner as a detailed graph. Entry is always to those zones for which external entry points are indicated. The only other distinction is that values are multi-attributed. Because of the conditions we have established for correct zone analysis, it follows that any navigation of a global graph induces (but not uniquely) a navigation on the corresponding detail graph. The paths of the induced navigation follow those of the given one, and merely add intra-zonal details along class B paths.



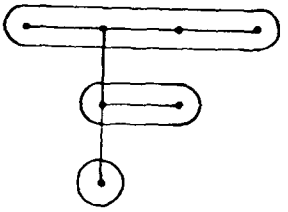
(Figure 2)



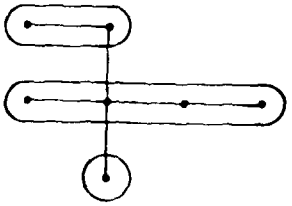
(Figure 3)



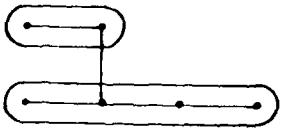
(Figure 5)



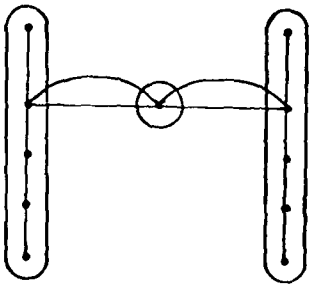
(Figure 7)



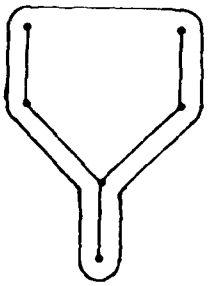
(Figure 9)



(Figure 10)



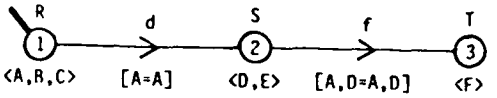
(Figure 11)



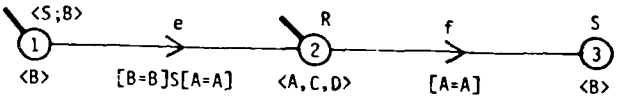
(Figure 16)

Zone Analysis for Seleicted Algebraic Graphs

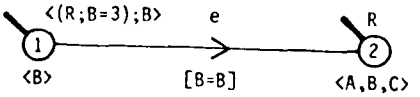
Figure 20



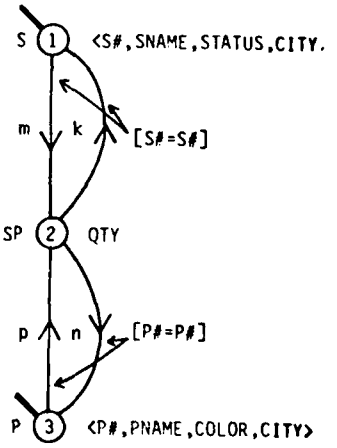

(Figure 7)




(Figure 9)




(Figure 10)

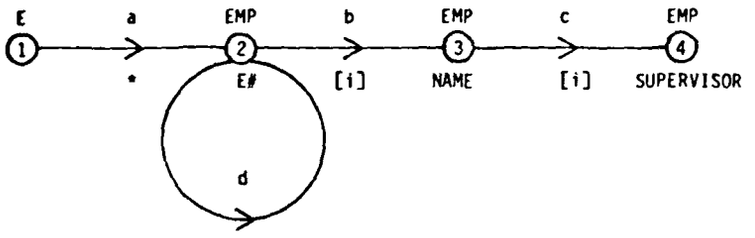



(Figure 11)

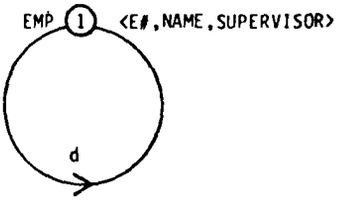
GLOBAL ALGEBRAIC ACCESS PATHS

FIGURE 21

EMP (E#,NAME,SUPERVISOR)

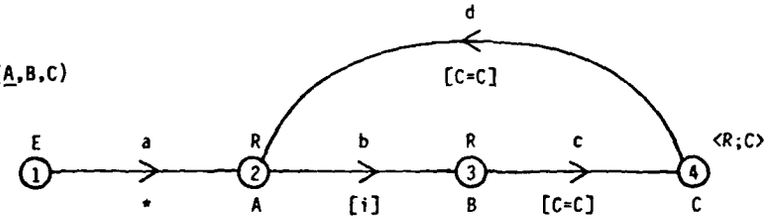



[SUPERVISOR=E#]




[SUPERVISOR=E#]

R(A,B,C)



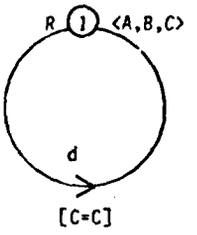




Figure 22

# 11.0 Application of Graph Theory to Algebraic Navigation

In this chapter, we present the application of some elementary graph theoretic results to algebraic graphs.

A digraph having n nodes can be represented as an nxn matrix by placing a 1 in a cell $C_{ij}$ to indicate a directed edge from $n_i$ to $n_j$, and a 0 in cell $c_{ij}$ otherwise. When this matrix is raised to the k-th power, a 1 in cell $c_{ij}$ indicates at least one path of length k from $n_i$ to $n_j$. (This matrix uses a Boolean arithmetic in which 1+1=1.)

This result can be extended to labeled digraphs of the type we have been using by placing in cell $C_{ij}$ the set of labels of all edges that traverse from $n_i$ to $n_j$, and putting the empty set in $c_{ij}$ if there are none. Multiplication of elements $c_{ij}$ and $c_{jk}$ is defined to be the set of pairwise concatentions of elements of $c_{ij}$ and $c_{jk}$ respectively. (Example: $c_{ij}$=(a,bc), $c_{jk}$=(de,fgh), and $c_{ij} \cdot c_{jk}$=(ade,afgh,bcde,bcfgh).) Addition is defined to be set union. When this matrix is raised to the k-th power, the cell $c_{ij}$ contains the set of all paths from $n_i$ to $n_j$ having length exactly k.

The previous result can be modified to develop retraction paths as well as direct paths as follows: Whenever the edge e appears in cell $c_{ij}$, the inverse edge $e^-$ must be placed in the conjugate cell $c_{ji}$. Multiplication may be performed as in the preceeding

paragraph, but some method must be chosen to remove spurious paths: (i) The matrix may be raised to the k-th power, and then all spurious paths removed from the result, or (ii) spurious paths may be removed after each individual cell multiplication. The second method involves less computation, but results in a matrix multiplication that is no longer associative but must be performed from left to right. The reason for this is that while any initial segment of a retraction path is a retraction path, this does not hold for terminal sections of retraction paths, or intermediate sections. The result of raising a matrix to the k-th power in this manner is a matrix in which the cell $c_{ij}$ contains the set of all paths and retraction paths from node i to node j of length exactly k.

Join conditions may now be added to this matrix, so that the individual element is not just an edge label, but a pair consisting of an edge label and a join condition. When the element $a/[jc_a]$ is multiplied by $b/[jc_b]$, the result is $ab/[jc_a]C[jc_b]$ where C is the control of node j, and the multiplication is between cells $c_{ij}$ and $c_{jk}$. This rule must be modified for retractions by (i) not adding on to the right end of the join condition when retraction edges are concatenated, and (ii) when the retracting is

ended and forward traversals begin, then the joining must be to the control of the node reached by the retraction rather than the right-most node of the string.

When a matrix is raised to the k-th power in this manner, the cell $c_{ij}$ contains a set of pairs, each of which is a path of length k from $n_i$ to $n_j$ and the join condition of that path. This join condition can be used to compute the effect of traversing the path.

## 12.0 The Hierarchy of Compiler Decisions

The following overview of query compilation treats the process as one of successive refinement of detail. Each step produces an enumeration of all possible ways to resolve a query at that level of detail. Each succeeding level elaborates the possibilities so as to produce an expanding tree of possible navigations. The evaluation of any one navigation for efficiency can only occur at a fairly low level. Certainly it is possible to apply some heuristics to this process, but this is tricky when one is operating in a perfectly general situation, and these may have to be developed in a somewhat application dependent manner. In any case, heuristics are premature until one has defined the space that is to be heuristically explored, and this in itself is quite a difficult task. The steps are as follows:

### 12.1 Navigation of a Global Graph

At this level of analysis, the query must be decomposed into those projections and/or restrictions of its relational variables that comprise the stored data. Therefore, all issues of geographic distribution must be resolved at this level, as well as all use of implemented joins among relations (including geographic collocation), and all "interesting" joins (class A) within an implementation. The existence of secondary indices may not be apparent at this point, depending upon the zone analysis used.

12.2 Navigation Within Zones

This step adds more detail, including secondary indices and traversals with each zone.

12.3 Access Ordering

The traversal of any edge which is one-to-many in the forward direction requires the access of all or a selected portion of a set of homogeneous elements. File access methods become apparent at this level.

Relational DIAM and DIAM I [7] requires that any one-to-many edge specify an ordering criteria which is always a function of the control of the destination node. Further elaboration is then expressed at the encoding level of the theory.

A useful approach to compilation would be to devise a set of commands such as get first, get next, get random and get sequence (ordered by the given ordering criteria). These operators correspond reasonably to the functional capabilities of existing systems. But one must be aware that some such request correspond to file access methods or link ("set") traversals between records, while others correspond to field selection within a record, therefore they cannot be interpreted uniformly. At least some encoding information is required to distinguish these.

The elaboration of a navigation at this level converts set operations into sequential operations controlled by loops as in computer programs.[5] The sequential dependencies resulting from the graph navigation have been preserved, making such programs applicable to database machines and networks having multiple access engines.

## 12.4 Evaluation of Navigation for Efficiency

Every proposed access strategy has given some consideration to this question. Schneider has used a complete DIAM extended even to the level of disk rotation speed and head position to simulate a databasse under given load conditions. Most compiler evaluation routines have modest goals, and are adequate for selection of a reasonable navigation. Yao [6] and others associate a cost with the traversal of each edge type. To make use of these costs, one has to have an estimate of the number of times an edge will be traversed in instance. An algebraic expression for this is always derrivable, therefore the problem becomes one of being able to estimate the cardinality of a relational query. Our own work on this subject is unpublished. There is a paper on the subject by R. Demolombe [9].

---

5 Some examples of these can be found in [8].

## 13.0 The Compilation of Navigations

The basic problem is that the structure (Global Graph Level) of a query and the structure of the database may be quite different, requiring that the query be decomposed and the components of the answer be reassembled. This paper has provided a detailed description of how to compute the query which is answered by a navigation. The inverse problem is to find a navigation (or all navigations) for a given query. Many investigators have described strategies for particular portions of this problem.

The only published literature to discuss in full detail the decomposition of a query in the context of all the possible distributions that can occur in a heterogeneous database network is [10]. This paper is obscurely published (fortunately soon to be made available through NTIS), and quite complicated. The modeling theory used is relational DIAM and its string model, but it is not difficult to restate in terms of algebraic graph theory. A brief outline follows:

1. The query is decomposed into one variable queries (See[11]) and all possible orderings of the variables are considered. (Some orders are equivalent to others for lack of dependencies.) Each one variable query specifies restriction conditions (which may be combined using conjunction and disjunction), is projected down to

only those attributes that are on the query target list and those needed for joins to other one variable queries, and makes use of information resolved by previous one variable queries for restrictive purposes.

2. Each one variable query must be covered by a set of stored projected restrictions (i.e. zone values) of its relational variable, such that the query can be constructed from the cover set using joins and unions.

3. Each zone may have several access routes, and any one of these may provide only partial access, requiring that they be used in combination. An access route may restrict based on joinability to another relation, because the path uses a join from another zone. This may limit the cover, or it may not if the same join appears in the original query.

4. Redundant components of the cover sets must be identified and removed.

5. Access paths must be examined for commonality, and common trunks replaced by grafts.

## 14.0 Conclusion

A general model for the logical connectivity of databases has been presented as a replacement for the relational DIAM string model. It has the advantages of simplicity, fidelity to the structure being modeled, universality, and the fact that navigations are easily defined and the information resolved by them readily computed. This theory can be used as the foundation for database access and database design research. More generally, it illustrates that there is a unifying science of database structure, and hopefully will inspire more research in that direction.

## REFERENCES

1. Codd, E.F. Extending the Database Relational Model to Capture More Meaning, ACM TODS, Vol. 4, No.4, December, 1979.

2. Senko, M.E. DIAM II: The Binary Infological Level and Its Database Language -- FORAL, IBM Research, Yorktown Hts., N.Y.

3. Schneider, L.S. A Relational View of the DIAM, Proc. ACM SIGMOD International Conference on the Management of Data, Washington, D.C., June 1976, pp. 75-90.

4. Smith, J.M. and Chang P.Y.T. Optimizing the Performance of a Relational Data Base Interface, 1975 SIGMOD Workshop, San Francisco, May, 1975.

5. Rothnie, J.B. et.al. Introduction to a System for Distributed Databases (SDD-1), ACM TODS, Vol. 5, No. 1, March 1980, pp. 1-17.

6. Yao, S.B. Optimizing of Query Evaluation Algorithms, ACM TODS, Vol. 4, No. 2, June, 1979.

7. Senko, M.E. A Data Independent Architectural Model -- Four Levels of Description, IBM Research, Yorktown Hts., N.Y., Report RJ982.

8. Levin, M. An Introduction to DIAM - Level of Abstraction in Accessing Information, ACM '78, Washington, D.C., December 1978.

9. Demolombe, R. Estimation of the Number of Tuples Satisfying a Query Expressed in Predicate Calculus Language, VLDB Conference, Montreal, 1980, pp. 55-63.

10. Schneider, L.S. A Relational Query Compiler for Distributed Heterogeneous Databases, SHARE 50 Proceedings, Denver, CO., March 1978.

11. Wong, E. and Youssefi, K. Decomposition-A Strategy for Query Processing, Memorandum No. ERL-M574, Electronics Research Laboratory, University of California, Berkely, CA, January, 1976.

VOLUME III

# DATABASE INVENTORY

CONTENTS

## 1.0 SUMMARY

In order to utilize the distributed - heterogeneous query compiler (or compare alternative candidate data base implementations,) a modeling technique must be employed which is independent of the data representation. Furthermore, the quantitative characteristics of the data population must also be described at the representation - independent level.

This paper discusses the terminology of the relational data model and describes how a relational model can be constructed from an existing collection of data; an approach to formulating a statistical description of the attribute values and tuple instances with respect to a relational schema; and a two dimensional approach to analyzing the data population as the data base analyst increases his or her understanding of the database.

Finally, the paper describes a set of seven forms which can be used as a convenient means of organizing the data being inventoried.

## 2.0 INTRODUCTION

A data-base management system (DBMS) is a generalized tool for manipulating large data bases. It is usually made available through special hardware and/or software for the interrogation, maintenance and analysis of data. The interfaces to a DBMS can provide a broad range of access to aid users from clerk to data administrator.

By now traditionally stated objectives of Data Base Management Systems are well documented and won't be cited here. However, the most evident trend visible in current thinking is that users of DBMS's are becoming increasingly oriented toward the information content of their data and less concerned with representation details. The user interface of a modern DBMS should deal with abstract information rather than the various lists, arrays, pointers, etc., which have so often been used to represent information.

Representation independence is even more significant when we consider the problems of information modeling on large data bases, the integration of existing data bases, and the requirements for simulation modeling of information systems for the purpose of evaluating and comparing candidate implementations.

This report presents a technique for describing both the qualitative and quantitative characteristics of a data base at a level that is independent of the implementation of the data base under

construction. We begin with a simplified discussion of Relational Models and how one would construct such a model from a collection of existing data. We then proceed to discuss some quantitative characteristics of the collected data. That is, we concern ourselves not only with what kinds of data we have collected but with how much. This Quantitative Data Description is a rather novel approach to formulating a statistical description of a data population, and can be a powerful tool to the data base designer. Furthermore, the quantitative characteristics of the data being analyzed are described at a level that is independent of the way the data is represented in the data base(s).

QDD is at once both a philosophy with which to approach a database inventory effort, and an analytic tool. It can provide insight into both the theoretical and pragmatic aspects of database design. It can also be used as a "cookbook" to develop a statistical view of a given data population.

# 3.0 DATABASE INVENTORY - QUALITATIVE DESCRIPTION

THE QUALITATIVE DESCRIPTION ANSWERS THE QUESTION, "WHAT INFORMATION IS STORED IN THIS DATABASE?"

We use the term "database to mean any repository of information. If we look at databases from the point of view of information management rather than computer technology, it makes little difference whether we are talking about a system with tapes and disks, a system of file cabinets and card indicies, or a wall map with a magnetic objects that can be moved around. These are all similar systems in that information is put into them to keep up with what is happening in the world, and information is accessed from them because someone needs to know.

The qualitative description tells us what information is stored in the database but not how it is stored. This is like having an inventory of all the books in a library but not having a description of where they are stored, in what order they are arranged on the shelves, or what card files or other aids there are to help in locating a book. Of course all these issues are important when it becomes necessary to locate a book, or catalogue a new book on arrival. But that will be addressed separately. The ability to describe what data is stored, but not how it is stored is called "representation-independence." Representation-independence is an extremely important concept if we wish to make an inventory of many

databases using different storage media, different software, different programming techniques, etc., and still be able to make meaningful comparisons. A specific example of the power of representation independence will be given as soon as there is sufficient material to work with (see Section 5).

The qualitative description tells us what information is stored but not how much of it. This is like having a description of what parts are stored in a warehouse, but not how many of each part. The question of how many is important if we want to design a warehouse and we need to know how big to make the bins and how wide to make the aisles to allow for traffic. Again, this is discussed separately under the heading "Quantitative Data Description" (QDD).

Excercise 1

The phone company publishes three directories (not including the yellow pages). One of them is used to locate a phone number and address when the name of the party is known, therefore the entries are ordered alphabetically by the name of party. The other two, which are less widely distributed, are ordered by phone number, and by street name and street address respectively. Do these directories contain different information or the same information but represented differently? If the latter, can you describe (in English)

what information these directories all contain, without biasing your description toward any one of them?

Excercise 2

Suppose that you have 30 nephews, nieces, aunts, uncles and cousins you feel obligated to send birthday greetings to. What are the relative advantages and disadvantages of keeping them all on a single list in the back of your appointment book vs. writing the name of each in the space provided for his/her birthday in the main portion of the appointment book? What about doing it both ways?

## 4.0 THE TYPE OF REPRESENTATION-INDEPENDENT QUALITATIVE INVENTORY THAT WE ARE USING IS CALLED A RELATIONAL SCHEMA

This is best illustrated by an example. Suppose that a very simple database is about planes. Each plane has a serial number that is different from the serial number of any other plane (e.g. AF30476), type (e.g. B52), a maximum speed, a location (given as a UTM grid number), a heading (compass direction), a velocity, and a time-and-date when all of the information was last reported. A relational schema for all of this is as follows:

Relation: PLANES

| Attribute | Domain |
|---|---|
| PLANE-ID | Plane identifier numbers |
| TYPE | Plane types |
| MAX-SPEED | Miles/hr |
| LOCATION | UTM grid number |
| HEADING | Compass direction |
| VELOCITY | Miles/hr |
| REPORTED | Time |

Figure 1

This schema tells us something about the type of data that is to be stored. It is not the data itself. The following diagram shows both the schema for the relation PLANES, and a sample of data.

Relation: PLANES

| PLANE-ID | TYPE | MAX-SPEED | LOC | HEADING | VELOCITY | REPORTED |
|---|---|---|---|---|---|---|
| AF32133 | B52 | 700 | 4220 | 143 | 0 | 12:22 |
| AF04271 | F8 | 2000 | 6213 | 212 | 800 | 15:32 |
| AF06111 | F8 | 2000 | 2413 | 72 | 450 | 14:49 |

Figure 2

Each row of data is called a "tuple," in this case, a 7-tuple. Each data element of a tuple is called an "attribute." At the head of each column of attributes is a "role name" which is part of the schema. The role name is usually chosen so as to be mnemonic of the meaning or role that the attribute plays in the relation. It is also necessary to specify the domain of possible data elements that underlie each attribute. This is done in Figure 1. For example, the attribute SERIAL-NO must take on values from the set of possible serial numbers of planes. Both VELOCITY and MAX-SPEED have the same underlying domain which is miles per hour. But the role names are different. So, all the role names of given relation must be different, but several of them may have the same domains.

The schema gives us no indication of how many tuples (rows) of data there are, and in fact, this will always be changing whereas

the schema is somewhat stable. The schema also does not suggest any way of ordering the rows because this would be describing how the data is stored rather than what it is.

What does the schema tell us? It tells us what type of data the database is designed to accept, and what type of questions can be asked of it. Given the schema of Figure 1, the following are meaningful questions:

1. What is the present location of AF84232, and when was this reported?

2. List the serial numbers of all planes that have a maximum speed of over 1000.

3. List the serial numbers of any B52's whose location is 4201-4255, 4301-4355, or 4423.

## 5.0 AN EXAMPLE OF A REPRESENTATION INDEPENDENT DATABASE INVENTORY

Suppose one finds a database with two kinds of records which look like this:

```
01  PLANES RECORD

    05  PLANE-ID
    05  LOC
    05  VELOCITY
    05  HEADING
    05  CREW OCCURS X TIMES
        10  SERIAL-NO
        10  ASSIGNMENT

01  PERSONNEL RECORD

    05  SERIAL-NO
    05  NAME
    05  RANK
    05  AGE
```

Figure 3

The database has two files, one for each type of record. Each file is ordered sequentially and indexed for rapid access, the first being ordered by PLANE-ID, and the second by SERIAL-NO. This database can be used to maintain the assignment of personnel to planes. Each plane crew consists of several crew assignments each of which has a name such as CO-PILOT or NAVIGATOR. The PLANES record has a repeating group that allows room for each assignment,

and the serial number of the person who fulfills that assignment at the present time.

Now consider a somewhat different database. It has three types of records:

```
01  PLANES RECORD

    05  PLANE-ID
    05  LOC
    05  VELOCITY
    05  HEADING

01  PERSONNEL RECORD

    05  SERIAL-NO
    05  NAME
    05  RANK
    05  AGE

01  CREW RECORD

    05  ASSIGNMENT
```

Figure 4

These records are arranged such that the first two can be retrieved by hash-coding on PLANE-ID or SERIAL-NO respectively, but the third type record has no direct means of access. The system allows records to contain chained pointers to other records to form what CODASYL calls "sets." The CREW record is a member record of two sets, and the parents of the two sets are the PLANES records

and the PERSONNEL records respectively. Although the CREW record contains only the CREW field, it always has two owners one of which is a PLANES record and the other a PERSONNEL record.

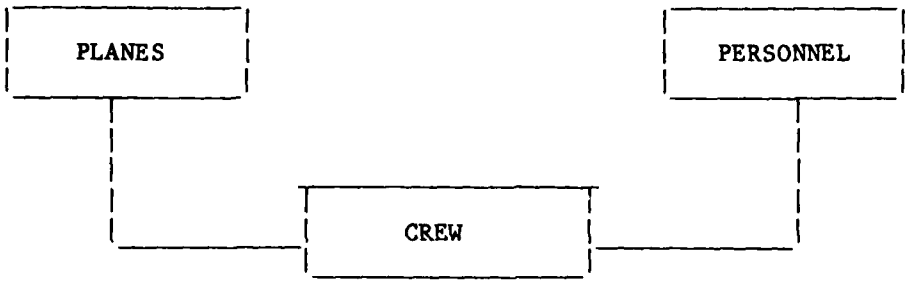



Figure 5

If you study these two database descriptions carefully, you will conclude that they both contain exactly the same information, although they differ considerably in organization and ease of access. For example, if you want to know the current assignment of a person given his serial number, there is a direct way to find this out in the second database, but in the first database it would be necessary to scan the PLANES file linearly until you found the assignment you were looking for.

A relational schema which models the information content of either of these databases, while not modeling the representation methods which make them differ considerably in appearance follows:

| | | | |
|---|---|---|---|
| Relation: | PLANES | Attributes: | PLANE-ID |
| | | | LOC |
| | | | VELOCITY |
| | | | HEADING |
| Relation: | PERSONNEL | Attributes: | SERIAL-NO |
| | | | NAME |
| | | | RANK |
| | | | AGE |
| Relation: | CREW | Attributes: | PLANE-ID |
| | | | SERIAL-NO |
| | | | ASSIGNMENT |

Figure 6

In this schema, we have specified role names but not the underlying domains as a matter of convenience. But they are important, and you should be able to specify them if you understand what this schema is about.

## 6.0 A KEY OF A RELATION IS AN ATTRIBUTE OR SET OF ATTRIBUTES WHICH UNIQUELY IDENTIFIES EACH TUPLE

Another way of saying this is that there can never be two or more tuples which are different but have the same key. In the example schema of Section 5.0, the PLANES relation has an attribute PLANES-ID which is a key. There is only one tuple for any plane in this relation, and each plane has a different PLANE-ID.

Very often, keys are created by database systems for the purpose of providing unique identification. A social security number is supposed to be a key (although there may be a few foul-ups) whereas a person's name is not reliable as a key. When a relation has a key which is a single attribute, then the relation generally contains facts about the object or person that the key stands for.

Relations having a key consisting of two attributes are frequently created to represent entities or events that have no name themselves, but can be named by means of several constituents. The CREW relation of the previous example may not have a single attribute key. If we assume that on any particular plane the crew assignment names are unique (i.e. a plane cannot have two CO-PILOTS) then the PLANE-ID and ASSIGNMENT jointly form a key. If we assume that no person is assigned to more then one CREW at a time, then the SERIAL-NO (of that person) is a key. If both of the preceeding

assumptions are true, then PLANE-ID and ASSIGNMENT form a key, and SERIAL-NO is a different key. So a relation may have more than one key.

From now on, all relational schemas will specify the keys of their relations. So:

| | | | |
|---|---|---|---|
| Relation: | PLANES | Attributes: | PLANE-ID |
| | | | LOC |
| Key: | PLANE-ID | | VELOCITY |
| | | | HEADING |

--

| | | | |
|---|---|---|---|
| Relation: | PERSONNEL | Attributes: | SERIAL-NO |
| | | | NAME |
| Key: | SERIAL-NO | | RANK |
| | | | AGE |

--

| | | | |
|---|---|---|---|
| Relation: | CREW | Attributes: | PLANE-ID |
| | | | SERIAL-NO |
| Key: | PLANE-ID, CREW | | CREW |
| Key: | SERIAL-NO | | |

Figure 7

Excerises

1. If we assume that a plane may have more than one of certain assignments (i.e. 2 CO-PILOTS), and if we assume that a person may

be assigned to several planes, but not more than one CREW position on any given plane, then what key or keys are there for CREW relation?

2. Try to invent a key for a relation pertinent to an airline reservation. Such a reservation guarantees to a particular person seating on a particular flight on a particular day. Since airlines do not request social security numbers or otherwise provide numbered accounts for customers, the customer's name will have to be used. What complications can arise from this? How might the key differ if the database was for one airline or for several.

7.0 IN A RELATIONAL SCHEMA, ALL INFORMATION IS EXPLICITLY DESCRIBED BY ATTRIBUTES OF RELATIONS, BUT IN STORED DATA STRUCTURES, INFORMATION IS FREQUENTLY CODED BY OTHER MEANS.

In Figure 3, there is a repeating group called CREW which contains the serial numbers and assignments of the crew. The crew of which plane? We can answer this by observing which PLANES record the repeating group is a part. In the relational schema (Figure 6 or 7) the CREW relation is viewed as a bundle of tuples, and the PLANES relation is viewed as another bundle of tuples. The only way to correlate a crew tuple with a PLANES tuple is by means of the common PLANE-ID attribute. Positional encoding in the file structure (Figure 3) has been replaced by specific attribute listing in the relational schema. Of course one might object that file structure of Figure 3 is a more useful way of representing this data. We might agree, but the point is that a relational schema is not intended as an implementation, but as an implementation-free method of describing the information content of implementations.

Regarding the file structure of Figure 4 and 5, we see a CREW record which has only the CREW assignment. How do we know what plane it refers to and what person? Because there is a system of address pointers that gets us from PLANES or PERSONNEL records to CREW records. Here, information which was made explicit in the relational model is encoded by way of address pointers.

Exercise

Consider a database consisting of a map of the world mounted on a cork backing, with different colored pins to represent various types of ships. Describe the techniques used to represent information. Make a relational schema for this information.

8.0 A "GOOD" RELATION REPRESENTS A FACT OR A COLLECTION OF FACTS ALL OF WHICH ARE ABOUT THE SAME ENTITY OR EVENT.

A "bad" relation can get us in trouble in several ways.

Consider the relation:

| | | | |
|---|---|---|---|
| Relation: | PLANES | Attributes: | PLANE-ID |
| | | | TYPE |
| Key: | PLANE-ID | | MAX-SPEED |

Suppose that MAX-SPEED is the same for all planes of the same type, e.g. the maximum speed of all B52s is 700. This relation is not "good" because it is supposed to be about individual planes identified by their IDs, but maximum speed is a fact about types of planes rather then individual planes. Consider some instance data:

Relation: PLANES

| PLANE-ID | TYPE | MAX-SPEED |
|---|---|---|
| AF04217 | F8 | 2000 |
| AF06111 | F8 | 2000 |
| AF32122 | B52 | 700 |
| AF34996 | B52 | 700 |
| AF33657 | B52 | 700 |

Figure 8

(i) The information concerning maximum speed is redundant. The speed of a B52 is represented 3 times above. (ii) If planes AF04271 and AF06111 were removed from this database, then the maximum speed of F8s in general would be lost. (iii) The database has no place to record the maximum speed of the B26 unless there is at least one such that we wish to list. (iv) Suppose that the maximum speed of F8s must be reduced because a structural flaw was discovered. We would then have to make note of this fact in no less than 3 places for the above database.

The idea of "good" relations is to have exactly one place to put each fact, neither more nor less. In the academic relational literature, this issue is treated under the heading "normal form." The treatment is mathematical, using what are called functional and multi-valued dependencies. But good relations can be obtained without using mathematics by (i) making sure that the key of a relation specifies some entity, fact, event, etc. which can be defined and understood to exist or occur in the world, and (ii) making sure that each attribute which is not part of the key is really determined by the entity specified by the key, and not by some subpart of it or some entity described by attributes other than the key.

"Bad" relations can generally be broken down into "good" relations.

For example:

Relation: PLANES Attributes: PLANE-ID
TYPE
MAX-SPEED
SENSOR-TYPE
FREQUENCY
SENSITIVITY

This relation describes planes, and the sensors mounted on planes. The maximum speed of a plane depends on its type, while the frequency and pulse-width of a sensor depends on the sensor type. This is altogether too much for one relation. So:

Relation: PLANES-TYPE Attributes: PLANE-ID
TYPE
Key: PLANE-ID

--

Relation: PLANE-SPEED Attributes: TYPE
MAX-SPEED
Key: TYPE

--

Relation: PLATFORMS Attributes: PLANE-ID
SENSOR-TYPE
Key: PLANE-ID
SENSOR-TYPE

--

Relation: SENSORS Attributes: SENSOR-TYPE
FREQUENCY
Key: SENSOR-TYPE SENSITIVITY

--

**Figure 9**

Some instance data may make it clear why breaking a single relation down into 4 smaller ones is desirable:

Relation: PLANES (first version)

| PLANE-ID | TYPE | MAX-SPEED | SENSOR-TYPE | FREQUENCY | SENSITIVITY |
|---|---|---|---|---|---|
| AF04271 | F8 | 2000 | X23 | 215 | .02 |
| AF06111 | F8 | 2000 | X29 | 340 | .06 |
| AF32122 | B52 | 700 | X23 | 215 | .02 |
| AF32122 | B52 | 700 | X24 | 620 | .005 |
| AF34996 | B52 | 700 | - | - | - |

Figure 10

The redundancies disappear when this is broken down into four smaller relations.

Relation: PLANES-TYPE

| PLANE-ID | TYPE |
|---|---|
| AF04271 | F8 |
| AF06111 | F8 |
| AF32122 | B52 |
| AF34996 | B52 |

Relation: PLANE-SPEED

| TYPE | MAX-SPEED |
|---|---|
| F8 | 2000 |
| B52 | 700 |

Relation: PLATFORMS

| PLANE-ID | SENSOR-TYPE |
|---|---|
| AF04271 | X23 |
| AF06111 | X29 |
| AF32122 | X23 |
| AF32122 | X24 |

Relation: SENORS

| SENSOR-TYPE | FREQUENCY | SENSITIVITY |
|---|---|---|
| X23 | 215 | .02 |
| X24 | 620 | .005 |
| X28 | 340 | .06 |

Figure 11

## Exercise

What is the problem with the following relation? Correct it.

Relation: CREW

Key: PLANE-ID, CREW

Key: SERIAL-NO

Attributes: PLANE-ID
TYPE
CREW
SERIAL-NO

## 9.0 ATTRIBUTES OF RELATIONS ARE SINGLE VALUED RATHER THAN BEING SETS OF VALUES

This means that no tuple of a relation may have more than one value at any given time. An attribute cannot be multi-valued like a repeating group or CODASYL "set." This does not present any problem in making a relational model of such a storage structure, but as you may have already found out, it is necessary to find the correct keys of the relations involved.

Suppose that we are examining a database that keeps track of the locations of planes. For each UTM, there is a list of the planes within the area. There could be no planes, one plane, or several. But each plane is in only one location (UTM). To represent this relationally, we use a single relation having two attributes, PLANE-ID, and LOC. Because each plane can be in only one location, PLANE-ID is the key of this relation.

Now consider a database relating parts of planes to planes. Each part is identified by a PART-NO. Any plane has many parts, and a part may be used on many planes. We also want to keep track of how many of each part are on each plane. (Please note that PLANE-ID specifies a particular plane not a type of plane, while PART-NO identifies a type of part but not an individual item. This distinction must always be understood and handled correctly.)

The relation:

| | | | |
|---|---|---|---|
| Relation: | PARTS | Attributes: | PLANE-ID |
| | | | PART-NO |
| Key: | PLANE ID, PART-NO | | QUANTITY |

has a two-attribute key because the relation between parts and planes is many-to-many. If we were to put each individual fact on a file card (e.g. PLANE-ID: AF04271, PART-NO: P586793, QUANTITY: 12), how could we arrange the file cards? Either by PLANE-ID and then by PART-NO or visa-versa. Either one might be useful, and either one could have disadvantages. Neither method is very convenient for both listing all the parts belonging to a plane and all the planes that use a given part. So we might want to use both of them or else make use of an index.

The point of this discussion is that although only one value of an attribute can go in a tuple, there can be many tuples to represent a multi-valued attribute. A relation between two entities may be one-to-one, many-to-one or many-to-many. In the first case, each entity alone is a key, in the second case the multi-valued attribute is the key, and in the third case both are a single key.

Sometimes, several facts are put together to form a single relation.

For example:

| | | | |
|---|---|---|---|
| Relation: | PERSONNEL | Attributes: | SERIAL-NO |
| | | | NAME |
| Key: | SERIAL-NO | | AGE |
| | | | NAME-OF-SPOUSE |

This works as long as each person in the PERSONNEL file has a single age, a single (complete) name, and at most one spouse. Each of these is a single valued fact about the same person. In a polygamous society, this relation is no longer workable. The multivalued attribute NAME-OF-SPOUSE requires a separate relation because SERIAL-NO no longer works as a key. So:

| | | | |
|---|---|---|---|
| Relation: | PERSONNEL | Attributes: | SERIAL-NO |
| | | | NAME |
| Key: | SERIAL-NO | | AGE |

--

| | | | |
|---|---|---|---|
| Relation: | PERSONNEL-SPOUSE | Attributes: | SERIAL-NO |
| | | | NAME-OF-SPOUSE |
| Key: | SERIAL-NO, NAME-OF-SPOUSE | | |

Exercise

1. Make a relational schema that shows an engine inspection history for planes. Each plane has possibly several engines (Engines No.

1,2,...). For each engine there should be a history of inspections with dates and name of mechanic performing the inspection.

2. Consider the relation:

| | | | |
|---|---|---|---|
| Relation: | PLANES-CREW | Attributes: | PLANE-ID |
| | | | PILOT |
| Key: | PLANE-ID | | CO-PILOT |
| | | | NAVIGATOR |
| | | | GUNNER |

Modify it so that a plane can have two or more co-pilots, gunners, etc.

## 10.0 AN INDEX ALLOWS MORE RAPID ACCESS TO INFORMATION, BUT IT DOES NOT ADD NEW INFORMATION TO THE DATABASE

Therefore, it is generally not necessary to model an index when one is making a relational schema. Indicies are of several types, and we mention them briefly.

When records are arranged in sequential order according to some key, an index which lists the highest keyed record to occur on each block (or page) allows rapid access to any record by key without having to read sequentially. This is called an indexed sequential file. The index does not add information, and does not occur in the relational schema.

Another situation is when there is a need to index on some attribute which is not the attribute used to physically order records. For example, we may order PLANES records by PLANE-ID, but want an index by type, speed, location etc. These are called secondary indicies. They allow faster access when we need to get a plane with a specific attribute, but they also cost something to maintain since they must be updated for insertion, deletions and changes of the attribute that they index. Again, such indicies do not add information and are not modeled in a relational schema.

As an extreme example, let us consider the phone book discussed in Section 3.0. If we were to cut it up into tiny scraps each of

which contained one party name, with address and telephone number, and stir these up randomly, there would be no loss of information. If all this data were keyed into a computer system, it would be easy to sort and print a telephone directory of each of the three types discussed.

A library may index all of its books by author, title and subject. But while any number of attribute indicies are possible, only one physical ordering of the books is possible. Usually this is by Library of Congress number, Dewey decimal number, or author (for fiction). Again, none of this is part of the relational model, we just want to know what books there are.

The purpose of this chapter is to indicate what not to pay attention to.

Exercise

Suppose that PLANES records do not tell us the color that the planes are painted. Somewhere else, there is a list of colors, and for each color, a list of planes painted that color. Is this a secondary index? How does it get treated in a relational schema.

## 11.0 WHAT IS A DATA ELEMENT?

So far, we have paid very little attention to the domains underlying the attributes of relations. Programmers and analysts deal with such domains all the time. Typically, they are concerned with questions such as: How is it represented inside a computer? How much space does it take? How long is it? Or else: How is it represented in printed reports? What is the algorithm for translating from the internal to the external representation and visa versa? These issues should be noted, but relational modeling has very little to add here.

Our main concern is with the "real world" or enterprise being modeled. The set of values of an attribute is determined by something out there. It may be a discreet set of choices, or a measurement having a certain dimensionality and required accuracy, or an identifier or name to be copied exactly, or a piece of text. In making an inventory which is broader in perspective than a single system, it is obviously to our advantage to collect as much descriptive information about the real domains as we can get. This is more valuable than coded fields used in a particular system.

One other question that comes up: How big is a data item, or one particular piece of information? This question has no absolute answer, but it may have an answer relative to our purpose. A data

item is something that for our present purposes is not going to be analyzed into smaller parts. So if we have a large printed report, and for present purposes have no need to open it, but can refer to it by title, catalogue number, or a few other attributes or key words printed on the cover, then it is perfectly legitimate to consider the contents as a single attribute of a relation.

## 12.0 MODELING A DATABASE RELATIONALLY

If you understand at least partially the issues discussed previously, you are ready to start analyzing a database and making a relational schema for it. This has been done already and it really works. (See "DIAMS Pacer Analysis")

One can begin by examining record descriptions, or printed forms, and finding out what fields are on them and what they are used for. Usually a record or form will have repeating items on it. These must be taken apart in order to make a relational schema as was previously discussed. The key of a given relation may not be immediately evident. One has to look for information implied by contiguity and by pointers and represent these with explicit attributes as was explained in Section 7.0. Multi-valued relations between things must be recognized, and handled properly as discussed in Section 8.0.

Although record descriptions and forms are a good place to begin, they are never a complete source of the information that you will need. The relational model contains more semantics of your enterprise than can be found in these. A complete understanding of a database requires making a connection between things happening "out there" and changes occurring to the database. One must also understand who needs to know what and how the database represents

this information. Such an understanding is not easy to come by. It may be largely undocumented, and parts of it may be understood by manager, analysts, programmers, field workers and secretaries.

When you have a preliminary schema for some portion of a database, check it. Ask these questions:

(i) What does this data tell us about the enterprise?

(ii) What events in the enterprise cause the database to change?

(iii) Who needs to know this information? What access route does he need? (i.e. what does he come in knowing in order to find out something else.)

(iv) What events cause a tuple in this relation to be inserted or deleted? What events cause an attribute value to change? Who uses this attribute?

Answers to Exercises

Section 3.0

1. These directories all contain the same information. They all contain associations of names, addresses, and telephone number.

2. Writing this information on the appropriate dates probably makes it more visible to you at the apropriate time. But when it is time to copy this information to next year's appointment book, you would appreciate having them all listed in one place.

Section 6.0

1. Under these assumptions, the only key is SERIAL-NO, PLANES-ID.

2. NAME-OF-PARTY, FLIGHT-NO, DATE is a key if only one airline is involved. Otherwise we should add a fourth attribute AIRLINE because the flight number is not unique.

There could be confusion between people with the same name making exactly the same reservation. The only way to clarify this is to ask for additional attributes such as address or phone number.

Section 7.0

1. Relation: SHIP-LOCATION Attributes: SHIP TYPE
LATITUDE
Key: SHIP-TYPE LONGITUDE
LATITUDE
LONGITUDE

Section 8.0

1. There is a one-to-many relation between planes and crew, but TYPE is an attribute of plane only. It should be removed, and placed in a separate relation:

Relation: PLANE-TYPE Attributes: PLANE-ID
TYPE
Key: PLANE-ID

Section 9.0

1. Relation: PLANE INSPECTION Attributes: PLANE-ID
ENGINE-NO
Key: PLANE-ID INSPECTION-DATE
ENGINE-NO INSPECTOR
INSPECTION-DATE

The key is correct only if there is never more than one inspection on a single day.

2. One method is to have several relations, one for each position type:

Relation: PLANE-PILOT Attributes: PLANE-ID
SERIAL-NO

Key: PLANE-ID
SERIAL-NO

Another method is too use only a single relation and not code the possible crew assignment into the schema, as in Figure 6.

Section 10.0

1. This is not purely an index. It contains additional information. Color is now an attribute of planes and should be handled accordingly. Does each plane have one color or several?

# 13.0 DATABASE INVENTORY - QUANTITATIVE DESCRIPTION

THE QUANTITATIVE DESCRIPTION ANSWERS THE QUESTION, "HOW MUCH INFORMATION IS STORED IN THE DATA BASE?"

If we are going to inventory the database, we are interested not only in what kinds of data we have, but also in how much. This latter aspect, which we call Quantitative Data Description (QDD), is aimed at at formulating a statistical description of the attribute values and tuple instances with respect to a relational schema. Doing analyses of this type is not a new subject in data processing. We have been concerned for decades about such things as the number of records in a file, the selectivity of an index, etc. But QDD is different in that it, just as with the relational model, approaches these issues independently of representation. This has two distinct advantages:

1. The inventory can be used to support studies of a broad range of implementation alternatives.
2. Many of the dynamics that would normally be inextricably bound-up in the representation details can now be described independently.

The fidelity of a QDD can vary principally in two dimensions:

1. The degree of interdependencies considered.
2. The degree of dynamics considered.

Since both of these are really a result of how much we understand about the database under study, QDD uses a two-axis approach that

allows the analyst to progress gracefully in both dimensions as his understanding of the database matures. This is more appropriately illustrated graphically in Figure 12.

Fidelity Space of Quantitative Data Description

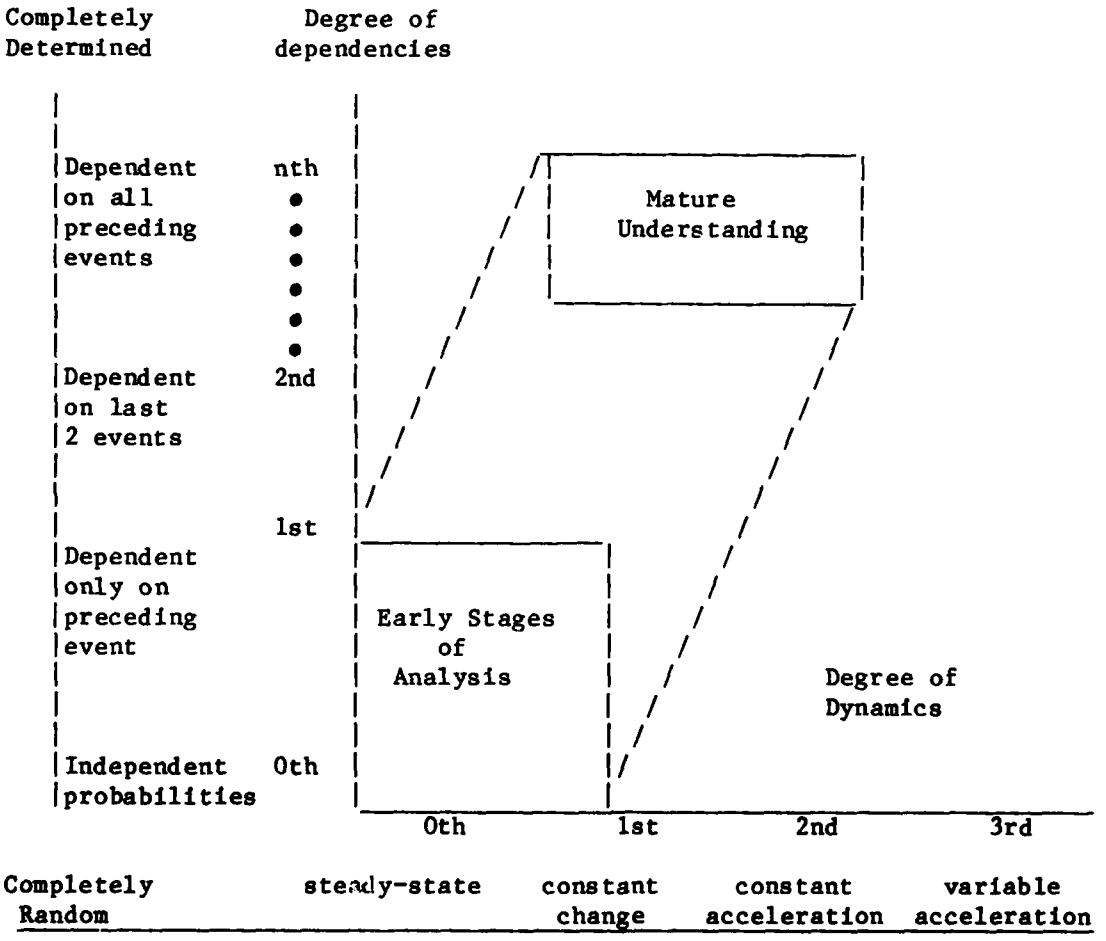



Figure 12

QDD is presented in the sections that follow in much the same way that the diagram indicates. Rather than approaching dynamics or dependence as separate subjects, QDD merely progresses from simple to complex.

## 14.0 THE SIMPLEST CASE

A relational model with one relation whose instances are uniformly distributed and never change is not a very likely occurrence and even if it were, it would probably not be the subject of a vigorous analysis. But unreal as it may be, it is a nice place from which to begin. In such a system, there are but three questions of a quantitative nature that we can ask:

1. How many tuples are in the relaiton?
2. How many values can each attribute assume?
3. For each attribute value, how many tuples in the relation contain that value?

The answers to 1 and 2 are, of course, just numbers and QDD uses the notation

$$\#(r)$$

to mean "the number of tuples in relation r" and

$$\#(a)$$

to mean the number of values of attribute a in relation r.

Since everything is uniform, question 3 doesn't really need an answer, but for reasons that will become obvious, we give it one anyway:

$$D(a,r)$$

denotes the distribution of tuples in r about the attribute a which

is, for the uniform case, simply

$$D(a,r) = \frac{\#(r)}{\#(a)}$$

For example, in the PLANES relation:

PLANES(ID,TYPE, UTM GRID NO., . . .)

#(PLANES) is the number of planes in the enterprise (e.g. 1,000); #(TYPE) is the number of plane type (e.g. 10); and D(TYPE, PLANES) is the average number of planes of a type which in the uniform case, is simply #(PLANES) divided by #(TYPE) = 100.

## 15.0 MULTIPLE RELATIONS

If there is more than one relation in the schema (or, even if there is only one but it has more than one attribute with the same underlying domain) we can no longer assume that an attribute value set and its underlying domain are equivalent. Specifically, we would like to know:

1. How much of the underlying domain is contained in the attributed value set?
2. How big is each domain?

Using the same notation, we can write:

$$\#(\text{PLANE-ID}) = 1500 \text{ where PLANE-ID underlies ID; and}$$

$$D(\text{PLANE-ID, ID}) = \frac{\#(\text{ID})}{\#(\text{PLANE-ID})} = \frac{2}{3} \quad .$$

Thus, if both PLANES and CREW have domains over the PLANE-ID domain; and

$$D(\text{PLANE-ID,PLANES.ID}) = .7$$

$$D(\text{PLANE-ID,CREW.PLANE-ID}) = .9$$

then, under the assumption of uniformity, we could compute the size of the intersection (using a binomial distribution).

## 16.0 NON-UNIFORMITY OF DISTRIBUTION

While uniform distributions occur frequently (and, unless we have specific knowledge to the contrary, uniformity is all we are justified in assuming) we call cite cases that we know for a fact that the PLANES population is actually divided by TYPE as follows:

Fighter = 550

Bomber = 250

Recon = 125

Transport = 50

Rescue = 25

This description of D(TYPE,PLANES) is obviously more accurate, but unfortunately, it is not necessarily more useful. Let's consider D(UTM-GRID-NO.,PLANES). It is not likely that this description is uniform since much of the airspace is restricted (there are clearly going to be more of our planes in the U.S. than in Eastern Europe). Now, suppose we are using a 1° UTM-GRID structure. There are going to be 360o360 = 129,600 grids (i.e., #(UTM-GRID-NO.) = 129,600). Not only will the table representing this distribution be rather large, it will also, in a sense <u>be</u> the database iteself. For if we did this for every attribute and every relation, we would have at least as much data as were in the database (unsorted of course) and, if the distributions were sparse as with UTM grids, we would

probably have more! This is definitely not the goal of QDD; we want to describe the values statistically -- not list them individually. But if we don't have the values, how can we distinguish among them to represent non-uniformity when it occurs?

## 17.0 RANK AND DENSITY OF RELATIONS

There is one distinguishing characteristic among attribute values besides the value itself -- namely, how many tuples in the relation do, in fact, contain each value. This may sound a little bit circular, but consider the TYPE example. Suppose we abstract the list of values by replacing each with a number that represents its rank (order) with respect to the number of tuples that contain it (most tuples is rank 1). We could then build the bar graph in Figure 13.

Rank vs. Density Chart for D(TYPE,PLANES)

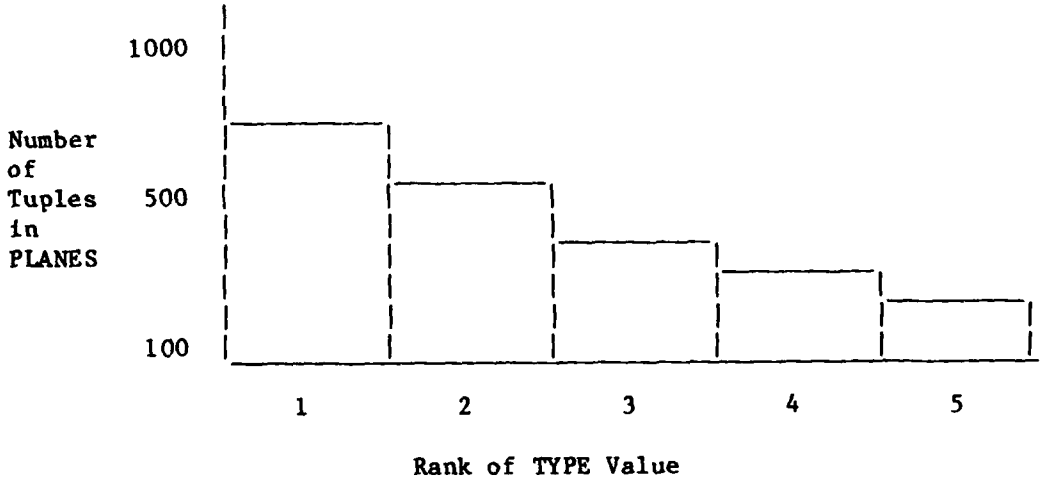



Figure 13

For quantitative analysis, this works as well as the table. We don't really care which TYPE is found in 550 tuples; just that such a TYPE exists. Furthermore, if we normalize the axes of this chart to:

Y: percentage of #(PLANES); and

X: percentile of #(TYPE);

we can handle large attributes such as UTM-GRID-NO. with relative ease as illustrated in Figure 14.

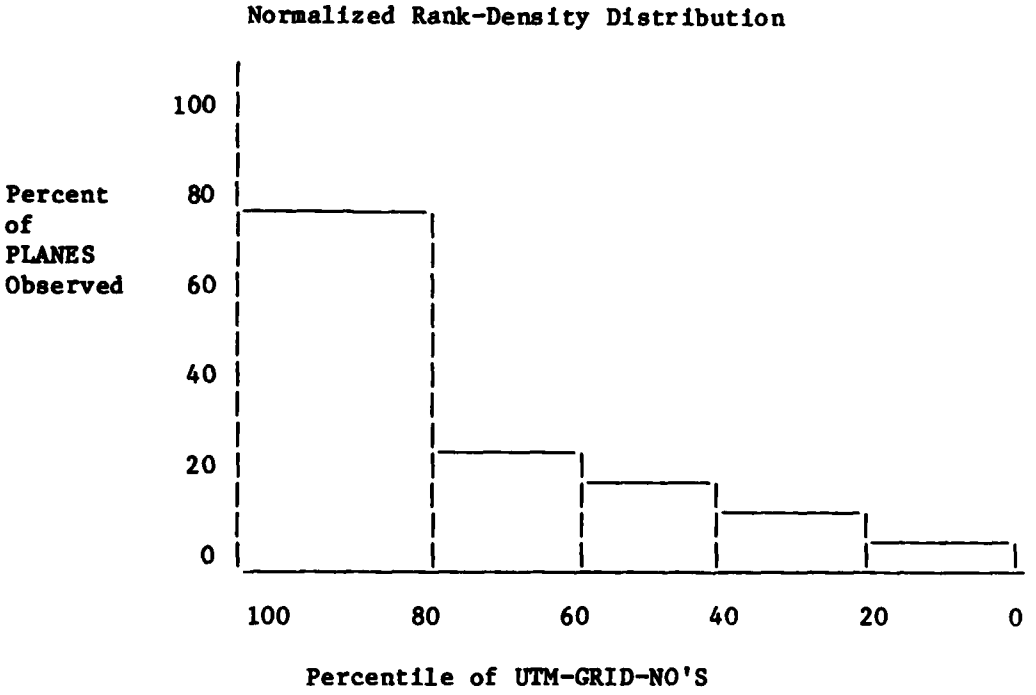
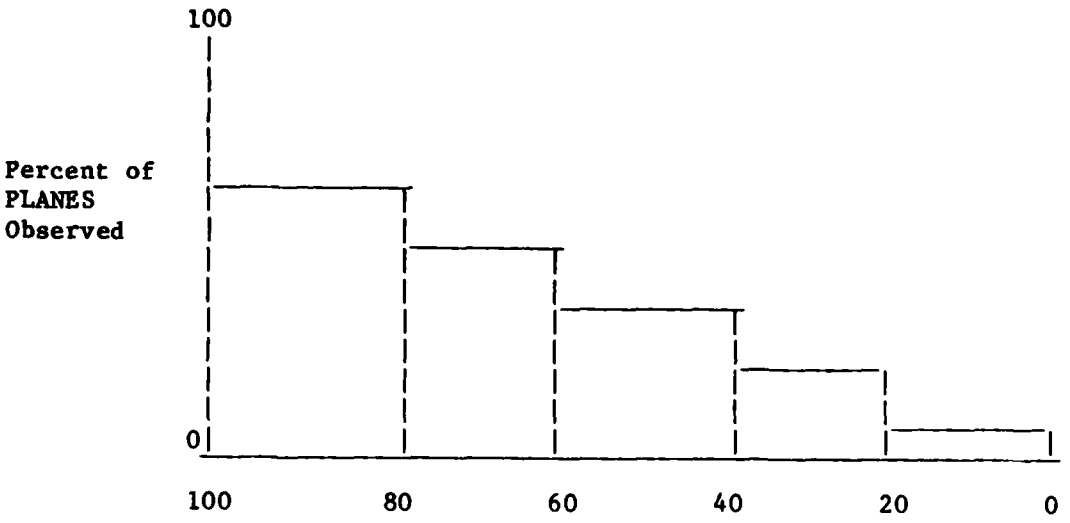



Figure 14

We can now, for example without worrying about how big #(UTM-GRID-NO) is, easily record that 80% of the PLANES are located in only 20% of the UTM-GRID-NO'S.

This same approach can be used to describe non-uniform distributions of attributes in their domains; e.g.




The only difference will be that the percentages will not sum to 100 because not all of the domain is observed in the attribute value set.

## 18.0 INDEPENDENT RATES

There is probably nothing in the world that happens independently, but when describing a model of the world, it is useful to distinguish between:

1. Exogenous Events - Those that occur outside of the model's structure (we will call these independent).

2. Endogenous Events - Those that occur as the result of an exogenous event because of the model's structure (we will call these dependent).

Since the relational model's link to real-world entities of interest is via the relations (rather than attributes), it stands to reason that the only objects of independent events are the tuples of a relation. They arrive, depart, change and get looked at because the entities they represent do so. It is therefore of primary importance that we describe the rate at which these independent events occur.

We make the following notation for an independent rate:

R(e,r) = t where

e is the event ("A"rrival, "D"eparture, "C"hange, "R"etrieval)

r is the name of the relation

t is the mean inter-event time

The use of inter-event times is a notational convention.

Thus, if our context is Griffis AFB and new planes are assigned there on the average of 12 times per week, we would describe this fact by the following notation:

$$R(A, PLANES) = 14 \text{ hours}.$$

This simple dynamic description forms the basis for describing some rather complicated events.

## 19.0 FIRST ORDER DEPENDENT PROBABILITIES - ATTRIBUTES

There are many aspects of dynamics that do not occur independently, particularly in the relational model. Specifically, these are the events involving attributes; i.e., nothing can happen to an attribute in a relation unless something happens to the tuple of which it is a member. Rather than describing these as a function of the independent rates, we find it more convenient (as will become apparent) to use the notion of a conditional probability.

For every attribute, a, of a relation, r, there exist a set of conditional probabilities of the form:

$P(e,a, \mid e_1,r)$ where

e is the event

a is the attribute

$e_1$, is the event with respect to the relation r.

Using this notation, we can describe the attribute dynamics in much the same way as those pertaining to the relations.

For example, returning to

PLANES (ID,TYPE,UTM-GRID-NO., . . .)

it is obvious, since ID is a Key, that

$$P(A,ID \mid A,PLANES) = 1$$

Which translates to say that if a PLANE tuple is added, it is certainty (P=1) that a new value of ID will also be added.

Similarly, assuming our list of types is comprehensive, we would postulate that

$$P(A,ID \mid A,PLANES) \approx 0$$

i.e., if a new plane is assigned to us, it is very unlikely ($P \approx 0$) that it will be of a TYPE we don't already have. Finally, since the most likely characteristic of a plane to change is its location, we might write

$$P(C,UTM\text{-}GRID\text{-}NO \mid C,PLANES) = 8$$

i.e., 80% of the changes to PLANES tuples will cause a change in the UTM-GRID-NO. And, if half of our planes are in a relatively sparse airspace, we might also note that

$$P(A,UTM\text{-}GRID\text{-}NO \mid C,PLANES)\ (.8)\ (.5) = .4$$

$$P(D,UTM\text{-}GRID\text{-}NO \mid C,PLANES)\ (.8)\ (.5) = .4$$

which says that about half of the location changes will be movement into a previously unoccupied grid (i.e., a new UTM-GRID-NO value will be added) and out of a grid that is now vacant (i.e. a value of UTM-GRID-NO will depart).

## 20.0 FIRST ORDER DEPENDENT PROBABILITIES - RELATIONS

Using the same notation, we can describe the fact that relations as well as attributes can also be the object of a dependent event. Adding the relation

CREW(SER#,PLANE-ID, . . .)

to our example, we might observe that 40% of the new planes assigned come to use complete with a crew, i.e.,

$$P(A,CREW \mid A,PLANES) = .4$$

All we have done is to replace an attribute name with a relation name (these can always be made distinguishable by using the relation name prefix).

Combinations of these probabilities can describe processes wherein a single, independent event triggers a complex network of associated events.

## 21.0 FIRST ORDER DEPENDENT PROBABILITIES - QUANTIFIERS

Up until now, we have considered only the simple case of single-valued events, e.g., the arrival of one plane, the arrival of one crewman, etc. While this is quite acceptable for independent events for which we know the multiplicity (e.g., if two planes arrive every 14 hours, we will have normalized this by the use of an inter-event time of 7 hours), it is not acceptable for (1) dependent events of (2) independent events whose multiplicity is dependent.

> e.g., in the previous example involving PLANES and CREW, it is likely that several crewmen will be assigned with a single plane -- but how many?

Fortunately, we have already recorded the answer -- we did this when we described the distribution

$$D(\text{CREW, PLANE-ID})$$

We simply need to invoke this distribution by asserting the likelihood that crew additions will be quantified by planes:

$$P(Q,\text{PLANE-ID} \mid A, \text{CREW}) = .5$$

i.e., half of the crew assignments will be a result of (or dependent on) plane assignments. By the same token if 70% of all inquiries about planes are by location, then

$$P(Q,\text{UTM-GRID-NO} \mid R, \text{PLANES}) = .7$$

This is an example of an independent event whose multiplicity is

dependent. If we wish, we can assert constant quantifiers such as

$$P(Q,6 \mid A,PLANES)$$

meaning of course, that planes are assigned six at a time (say, by squadron). But an even more useful aspect of constant quantification is exemplified by

$$P(Q,4 \mid Q,UTM\text{-}GRID\text{-}NO) = .5$$

This might represent the fact that half of the quantification by location is done in 2° arcs even though the resolution of the grid is 1°.

With this in hand, we could <u>begin</u> to describe a rather complex scenario such as

> "half of the times that a plane enters a new grid, the ID's of all other planes in the grid are displayed and, in 20% of the cases, so are the ID's of all planes in adjacent grids. This is the <u>only</u> case in which planes are displayed by location."

$$P_1(C,UTM\text{-}GRID\text{-}NO. \mid C,PLANES) = .8$$

$$P_2(R,PLANES \mid C,PLANES) = (.8)\ (.5) = .4$$

$$P_3(Q,UTM\text{-}GRID\text{-}NO. \mid R,PLANES) = (.8)\ (.5)\ (1) = .4$$

$$P_4(Q,9 \mid Q,UTM\text{-}GRID\text{-}NO) = (.4)\ (.2) = .08$$

We say "begin" because we cannot, as yet, describe the fact that

$$P_3(Q,UTM\text{-}GRID\text{-}NO \mid R,PLANES)$$

is either 0 or 1 depending on the occurrence of a two-event history.

## 22.0 SECOND-ORDER BIAS

In probability theory, the natural progression from first to higher order of dependence is simply the use of conditional probabilities with complex conditions. In the preceeding example, the precise statement should have been:

$$P_3(Q,\text{UTM-GRID-NO} \mid R,\text{PLANES} \wedge C,\text{PLANES}) = .4$$

and furthermore, that there is a set of probabilities, P, for (Q,UTM-GRID-NO) such that the sum of the probabilities in P is 1 and there is one element in P for each possible combination of events that affect (Q,UTM-GRID-NO). While QDD eventually achieves and even exceeds this precision, the use of complex conditional probabilities is avoided for two reasons:

1. Database analysis generally reveals dependencies on complex conditions one at a time -- long before the set of all possible combinations and their associated probabilities can be enumerated;
2. The potential number of combinations is staggering.

In order to fill this gap, which is essential if the graceful progression is to be maintained, QDD uses the concept of a <u>bias</u>.

The bias of event c on the probabilities of event a, written

$$B(a \mid c)$$

is the precentage by which the probability of event a increase or decrease given that event c has occurred.

Mathematically,

$$B(a \mid c) = \begin{cases} + \dfrac{P(a \mid c) - P(a)}{1 - P(a)} & \text{for} \quad P(a \mid c) \geq P(a) \\ - \dfrac{P(a) - P(a \mid c)}{P(a)} & \text{for} \quad P(a) \leq P(a \mid c) \end{cases}$$

with the nice result that

$$B(a \mid c \wedge d) = P(a) + B(a \mid c) + B(a \mid d)$$

which also commutes if all the biases are in the same direction.

## 23.0 SECOND-ORDER BIAS - ATTRIBUTES AND RELATIONS

Suppose that in the relation

PLANES(ID,RANGE,MAX SPEED,WEIGHT,. . .)

we observe that, although the RANGE, MAX SPEED and WEIGHT of a plane rarely change (say $P \approx .01$), if WEIGHT does change, both RANGE and MAX SPEED will surely be affected. Notionally, we have

$$B(C,RANGE \mid C,WEIGHT) = +100\%$$

$$B(C,\ MAX\ SPEED \mid C,WEIGHT) = +100\%$$

Thus even through normally

$$P(C,RANGE \mid C,PLANES) \quad .01$$

The occurrence of (C,WEIGHT) will increase this to 1.

This same sort of phenomenon is present among relations. Suppose we have the three relations: PLANES, SENSORS, PLATFORMS and we know that

$$R(R,PLANES) = 20\ minutes$$

$$R(R,SENSORS) = 20\ minutes$$

$$R(R,PLATFORMS) = 1\ week$$

Since PLATFORMS contains a foreign key to both PLANES and SENSORS, it is also reasonable that we might have

$$P_1(R,PLATFORMS \mid R,PLANES) = .5$$

$$P_2(R,PLATFORMS \mid R,SENSORS) = .5$$

But if a query involves both PLANES and SENSORS, it is reasonable to expect that the likelihood of retrieving PLATFORMS is increased over either $P_1$ or $P_2$. We can represent this by adding

$$B(R,PLATFORMS \mid R,PLANES) = .5$$

$$B(R,PLATFORMS \mid R,SENSORS) = .5$$

The result would be that if both PLANES and SENSORS are retrieved, the probability of PLATFORMS being retrieved would increase to

$$.5 + .5\ (1 - .5) = .75$$

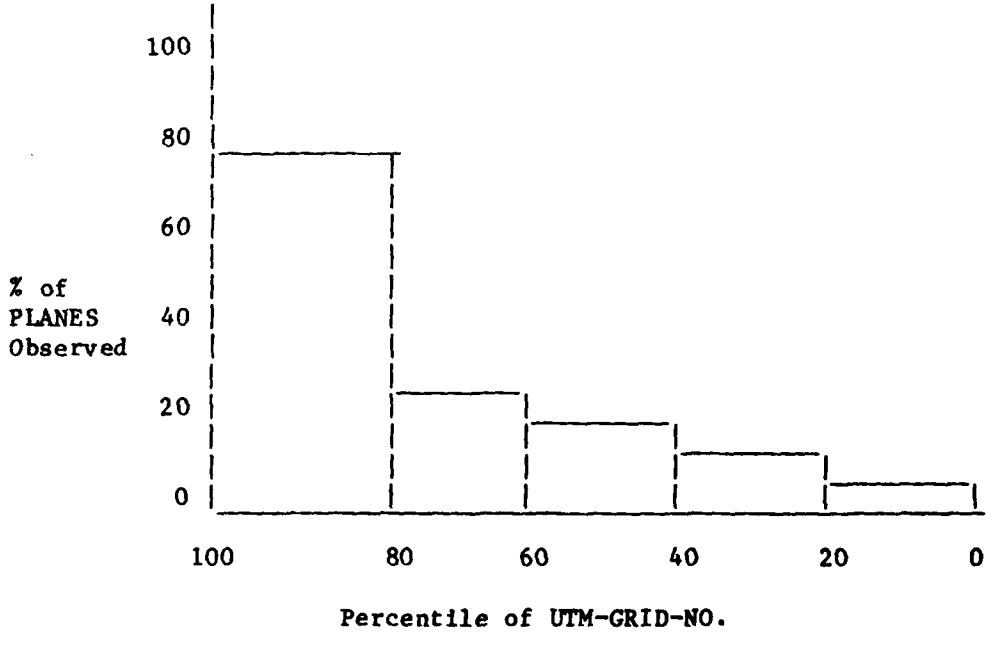
## 24.0 SECOND ORDER BIAS - QUANTIFICATION

There are many ways in which biases show up in quantification. The most obvious of these is an analogy of the attribute bias; e.g.,

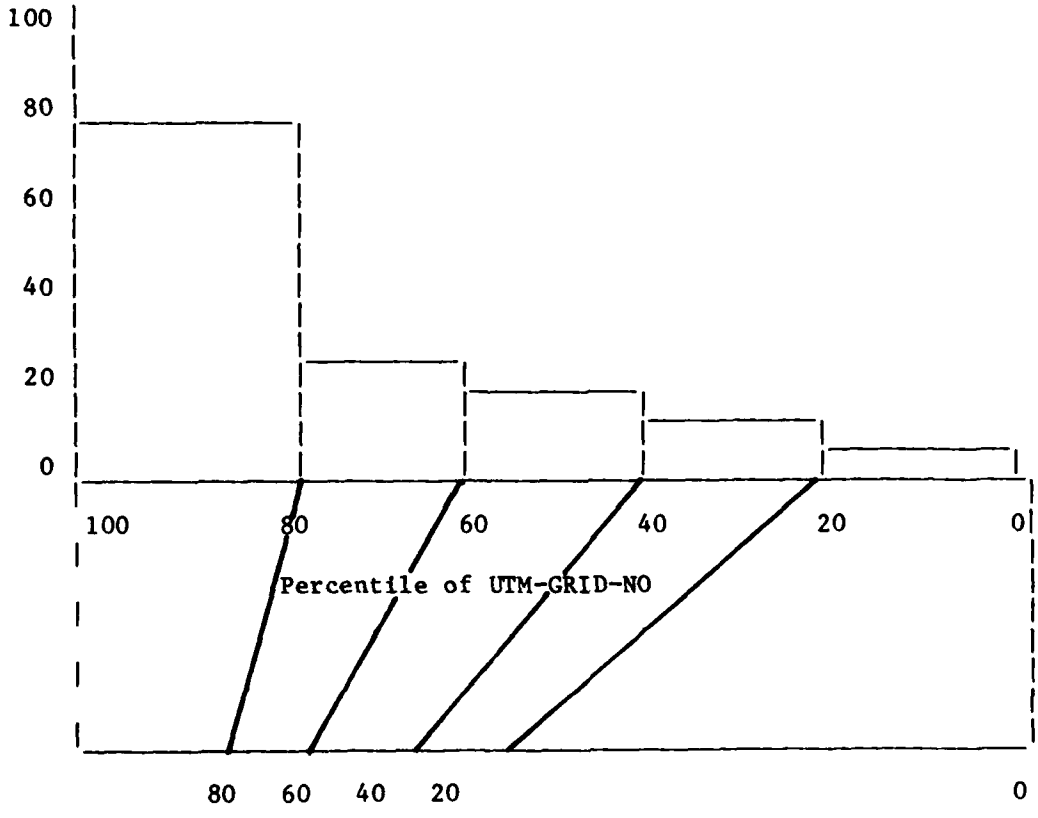
$$B(Q,\text{MAX SPEED} \mid Q,\text{RANGE}) = .8$$

meaning that if a PLANES event is quantified by RANGE, the likelihood that it will also be quantified by MAX SPEED increases 80%. But there are two more important functions of biases in quantification involving the distributions.

The first of theses is concerned with non-uniform access and is best illustrated by example. Recalling the (PLANES,UTM-GRID-NO) distribution



Remeber that we hypothesized that this non-uniformity of the distribution was due to the existence of restricted airspace -- i.e., there are grids that are usually empty because we do not allow our planes to go there. But by the same token, if we are monitoring the airspace for violations, then it is precisely these grids that we will be interrogating. We will be using the already non-uniform UTM-GRID-NO attribute non-uniformly to quantify planes as illustrated in the following diagram:




Percentile of UTM-GRID-NO biased by non-uniform use in quantification

That is, under uniform access, each percentile would have an equal probability of being accessed (.01). But if used for quantification, the probability of accessing the first 20 percentiles is greated than the sum of the probabilities for the other 80. Using the bias notation, we would write

$$B(X,\text{UTM-GRID-NO} \mid Q,\text{UTM-GRID-NO}) = V$$

where $V = (b_1, \ldots, b_n)$

n = number of division in the percentile axis to describe a bias to the X axis of the distribution. Thus, the example would be described by the statement

$$B(X,\text{UTM-GRID-NO} \mid Q,\text{UTM-GRID-NO}) = (-.8-.6-.5-.5+.7)$$

The other important function of biases to the distribution is to describe covariance of quantifiers. This, too, is best illustrated by example, this time a highly simplified version of

PLANES (ID,TYPE,MAX SPEED, . . .)

where there are only two TYPE's (fighter, transport) and two MAX SPEED's (fast, slow). Supposing that #(PLANES) = 1000 and both distributions are uniform, we would expect that

$$D_1(\text{TYPE},\text{PLANES}) = 500$$

$$D_2(\text{MAX SPEE},\text{PLANES}) = 500$$

Now, let's further suppose that we are interested in the complex quantification of PLANES by TYPE and MAX SPEED. If we have only $D_1$ and $D_2$, both of which are uniform, we would expect the composite distribution to be a uniform 250. But, as common sense dictates, this is not the case, we obviously have a lot of fast fighters and a lot of slow transports, but not very many slow fighters or fast transports. Hence, the composite distribution we would like is extremely non-uniform because there is a significant covariance between a plane's type and its maximum speed.

Using the same notation, but this time to bias the Y axis of the distribution, we would write

$$B(Y,\text{TYPE} \mid Q,\text{MAX SPEED}) = (+.5, -.5)$$

$$B(Y,\text{MAX SPEED} \mid Q,\text{TYPE}) = (+.5, -.5)$$

which would cause the desired skew.

## 25.0 TOWARD THE DETERMINISTIC CASE

While a second-order dependent model as we describe can be extremely descriptive of the data base in the planning stages, we all know that as a system design matures, dependencies far beyond the second order become apparent. And though our descriptive techniques thus far have been well suited to highly stochastic situations, as we approach the deterministic case there are clearly more natural ways to proceed.

Since the analyst already has a familiar concept for describing the completely deterministic case (for it is equivalent to describing the system itself), it makes sense, at this point, to shift into this context, and then relax the deterministic assumptions until we achieve something of a continuum back to the second-order model. To do this, we first need to describe:

1. Representation-Independent Accessing Language (RIAL) statements,
2. Traffic Sources, and
3. A correlation between 1 and 2.

And while this kind of description is in no way novel, we present our particular conventions before proceeding.

The RIAL syntax is diagrammed in Figure 15 and the only semantic explanation that should be necessary is that, because we cannot recognize attribute values, <value> means "number of instances"

and can be either <positive interger> or "the number of instances in" <set name>. Using this syntax, we write named RIAL statements that we call Transaction Type Definitions (TTD's).

Our Source Definitions consits of simply a Source Name, a mean intermessage time, a start time and a duration.

The correlation which we call a Source/Transaction Profile (STP) then consists of a three dimensional matrix of Source Name ($SN_i$) vs. $TTD_j$ vs $TTD_k$ where each cell is defined to be the probability (assumed for now to either 0 or 1) os $SN_i$ transmitting $TTD_j$ given that the previous transmission from $SN_i$ was $TTD^k$.

Thus we have a conventional model of the deterministic representation-independent dynamics of a data base application defined in terms that should be comfortable for the analyst. What remains, then, is to relax the deterministic assumptions until we meet the second-order dependent case.

Relaxing the sequence of transaction is trivial -- we obviously allow probabilities between 0 and 1 in the correlation matrix. Making the TTD's non-deterministic is a bit more chanllenging. We accomplish this by an extension to the RIAL called the RIAL Statement Description (RSD) Language. The RSD syntax (also diagrammed in Figure 15) is very similar to the RIAL syntax except that random substitutes are available to denote any term in a RIAL statement

that we do not wish to predetermine. The presence of a substitute denotes that the term is derived probabilistically from the second-order model. Thus, after a second-order model has been developed, we can gradually begin making the QDD more precise by the inclusion of less and less probabilistic RIAL statement descriptions until the dynamics are eventually determined.

## RSD/RIAL Syntax Diagrams

| RIAL | RSD |
|---|---|
| <statement>::=<function>(<set name>OF<argument>) | - |
| <argument>::=<object>WHERE<qualifier> | ARG * <positive integer> |
| <object>::=<Relation name>(/<attribute name>) | OBJ * <positive integer> |
| <qualifier>::=<condition>(OR<condition>) | QLF * <positive integer> |
| <condition>::=<quality>(AND<quality>) | CND * <positive integer> |
| <quality>::=<attribute name>=<value> | QLT |
| <value>::=<set name>/<attribute name>\|<positive integer> | - |
| <Relation name>::=<name> | RN |
| <attribute name>::=<name> | AN |
| <set name>::=S<positive integer> | - |
| <name>::=(<character>) | - |
| <character>::=<positive integer>\|<alphabetic character> | - |
| <alphabetic character>::=A\|B\|C.....\|X\|Y\|Z | - |
| <positive integer>::=1\|2\|3..... | R |
| <functions>::=A\|D\|R\|C | F |

Figure 15

# 26.0 RIAL AND RSD

A RIAL clause has the following basic form:

<temporary relation> OF <object relation> WHERE <qual>

The predicates in <qual> may reference <temporary relation>'s formed by other clauses in the same query thus giving rise to the linking or multi-variable query. Because it employs dependent clauses rather than nesting, queries can be constructed a clause at a time in any order. The following hypothetical database and queries should provide enough of the flavor of RIAL for understanding its use in QDD.

Assume the example database about planes, the sensors they are carrying (e.g., cameras, pollution detectors, etc.), and the places they are flying. Involved are six relations;

SENSORS (TYPE,WEIGHT, FREQUENCY,PULSEWIDTH,SENSITIBITY,POWER-RQMT)

PLATFORMS(PLANE-ID,SENSOR,TYPE,DATE INSTALLED)

PLANES(ID,NAME,TYPE,MAX SPEED, RANGE, UTM-GRID-NO,TIME REPORTED,
HEADING, VELOCITY)

ATLAS(GEOPOLITICAL NAME,UTM-GRID-NOS)

DISTANCES (UTM1,UTM2,KILOMETERS)

CREW(NAME,DUTY,PLANE-ID)

Platforms are the installations of sensors on a plane and planes vs. sensors in m:n. Universal Transverse Mercator (UTM) Grids are merely a division of the world into 1° squares and they are a useful and popular means for rough geographic correlation. Thus in the ATLAS, there will be as many tuples for a GEOPOLITICAL NAME as there are grids that the entity covers or partially intercepts. The DISTANCES relation is simply the center-to-center distance between and two grids and could be either a stored table or an algorithm as RIAL does not require the user to distinguish. We now present five examples; the first statement is the query in English, the second is its RIAL form and the third its RSD form:

Example 1:

"List the weight and types of sensors with a 3-ms pulsewidth and either 120- or 240-MHz frequency"

R S1 OF SENSORS/TYPE/WEIGHT WHERE PULSEWIDTH = 3 MS and FREQUENCY = (120, 240 MHZ)

R S1 OF SENSORS/TYPE/WEIGHT WHERE PULSEWIDTH = 1 and FREQUENCY = 2

Example 2:

"List the weights and types of sensors with a 6-ms pulsewidth and 120-MHz frequency or a 3-ms pulsewidth and 1.7 mV sensitivity"

```
R S1 OF SENSORS/TYPE/WEIGHT WHERE PULSEWIDTH = 6 MS AND
FREQUENCY = 120 MHZ or PULSEWIDTH = 3 MS AND SENSITIVITY =
1.7 mV

R S1 OF SENSORS/TYPE/WEIGHT WHERE PULSEWIDTH = 1 AND FREQUENCY
= 1 OR PULSEWIDTH = 1 AND SENSITIVITY = 1
```

Example 3:

"List the ID of all planes currently over Colorado"

```
R S1 OF PLANES/ID WHERE UTM GRID NO = S2/UTM GRID NOS
  S2 OF ATLAS/UTM GRID NOS WHERE GEOPOLITICAL NAME = COLORADO

R S1 OF PLANES/ID WHERE UTM GRID NO = S2/UTM GRID NOS
R S2 OF PLANES/UTM GRID NOS WHERE GEOPOLITICAL NAME = 1
```

Example 4:

"List the ID, type, and heading of all planes within 1000 km of Denver"

```
R S1 OF PLANES/ID/TYPE/HEADING WHERE UTM-GRID-NO = S3/UTM1
R S2 OF ATLAS/UTM GRID NOS WHERE GEOPOLITICAL NAME = DENVER
  S3 OF DISTANCES/UTM1 WHERE UTM2 = S2/UTM-GRID-NOS AND
        KILOMETERS ≤ 1000

R S1 OF PLANES/ID/TYPE/HEADING WHERE UTM-GRID-NOS S3/UTM1
R S2 OF ATLAS/UTM-GRID-NOS WHERE GEOPOLITICAL NAME = 1
R S3 OF DISTANCES/UTM1 WHERE UTM2 = S2/UTM-GRID-NOS AND
        KILOMETERS = 1000
```

Example 5:

"List the ID, type, heading, and velocity of planes over Colorado that are carrying sensors with a 3-ms pulsewidth, 240-MHz frequency, and a sensitivity 1.7 mV"

```
R S1 OF PLANES/ID/TYPE/HEADING/VELOCITY WHERE UTM-GRID-
        NO = S2/UTM GRID NOS AND ID = S3/PLANE ID
  S2 OF ATLAS/UTM-GRID-NOS WHERE GEOPOLITICAL NAME = COLORADO
  S3 OF PLATFORMS/PLANE-ID WHERE SENSOR TYPE = S4/TYPE
  S4 OF SENSORS/TYPE WHERE FREQUENCY = 240 MHZ AND PULSEWIDTH
        = 3 MS AND SENSITIVITY = 1.7 mV

R S1 OF PLANES/ID/TYPE/HEADING/VELOCITY WHERE UTM-GRID-NO
        = S2/UTM-GRID-NOS AND ID
R S2 OF ATLAS/UTM-GRID-NOS WHERE GEOPOLITICAL NAME = 1
R S3 OF PLATFORMS/PLANE ID WHERE SENSOR TYPE = S4/TYPE


R S4 OF SENSORS/TYPE WHERE FREQUENCY = 1 AND PULSEWIDTH = 1
        AND SENSITIVITY =1
```

Adds, Deletes and Changes are analagous with the following exceptions

(1) In an Add, the predicates not referring to other variables are interpreted as assertions for the attribute values and, therefore, in RSD must have the value 1.

(2) In a Change, the predicates that are over attributes that also appear in the object are interpreted as assertions for the new attribute values and must therefore have the value 1 in RSD.

For example:

```
A PLANES/ID, TYPE WHERE ID = 12345 AND TYPE = TRANSPORT
```

means to create the tuple <12345, TRANSPORT> in the PLANES relation

```
C PLANES/TYPE WHERE TYPE = FIGHTER AND MAX SPEED ≥ 1400
```

means to change the TYPE of all planes with MAX SPEED's over 1400 to "Fighter."

Using these examples we can now illustrate how RSD allows us to "back-off" from the strictly determined transaction and allow for those things we are not yet sure of.

Suppose in example 1 that, although we know that FREQUENCY will be in the qualifier, we're not yet sure how many values will be specified. We could use RSD to write

```
R S1 of SENSORS/TYPE/WEIGHT WHERE PULSEWIDTH = 1 AND
FREQUENCY = R.
```

This would permit any declarations such as

$$P(Q3 \mid Q,FREQUENCY) = .4$$

to govern our description with respect to the number of values of Frequency.

If we were less certain about the qualifier to the extent that we did not know what attribute in addition to PULSEWIDTH would be used in the qualifier but that there would be one, we would write.

```
... WHERE PULSEWIDTH = 1 AND QLT
```

This would allow the description to be governed by the set of:

$$P(Q,a \mid R, SENSORS)$$

where a is an attribute of SENSORS and the set

$$B(Q,a \mid Q,PULSEWIDTH).$$

Relaxing further, if we knew only that two attributes would be in the qualifier but not necessarily which ones, we would write

```
R S2 OF SENSORS/TYPE/WEIGHT WHERE QLT*2
```

There is obviously no limit to the degree of relaxation invokable, and even

STM

is a perfectly valid description, for reasons that will become apparent in considering sources.

Advancing to a multi-variable case, as we have in examples 3 & 4, we can take advantage of previously defined inter-relation probabilities. Suppose we do not know whether the query will involve distance, as in 4, or the atlas as in 3. We could rewrite 4 as

```
R S1 OF PLANES/ID/TYPE/HEADING WHERE UTM-GRID-NO = S2/UTM1
R S2 OF ARG
```

which would cause

P(R,r | R,PLANES) where r is a relation to govern.

Or, in example 5, suppose we knew that SENSORS would be involved but we didn't know whether the PLANES retrieval would cause a retrieval or deletion from SENSORS. We would rewrite S4 as

FNC S4 OF SENSORS....

allowing the set of

P(e,SENSORS | R,PLANES) where e is an event to govern.

## 27.0 WHERE DOES THE DATA ORIGNATE

Since the relational model does not involve the concept of source, its introduction is as much qualitative as it is quantitative. For this reason, we ought to begin with a discussion of: (1) What sources are; (2) Why we should be interested in Sources; (3) What information about Sources should be collected in the inventory process.

A source is any person or thing which generates events in the database. In an ideal situation, there is one and only one perfect source for each fact in the database. The failure of virtually all databases to achieve this ideal gives rise to many of the problems that database administrators must contend with.

> Integrity: If sources were perfect, i.e., never misperceived real world events, there would be no integrity problems in the database. But failures of people, sensors, communication links, etc., require that a system of checks and balances be invoked to prevent inaccurate data from entering the system. For example, if we're keeping track of roads and bridges, we should be very hesitant to admit a bridge observation if its endpoints don't correspond with a road.
>
> Concurrency: If each fact had only one source, there would be no concurrency problems. But again, the typical case does not conform. In fact, too often, a carefully designed relational model reveals how an enterprise should be organized as opposed to how it is but organizations are slow to change and the result is illogical, overlapping responsibilities and attendant confusion in the database.

Currency: By the same token, many facts we thought were being maintained turned out to have no source at all; i.e., a gap rather than an overlap in responsibility. In this case the result is self-deception rather than confusion. If we don't maintain a fact, or if we maintain it but at a slower rate than it changes in the real-world, then why waste space on it?

Utilization: We may also find that while a fact is being accurately maintained, it is not being used. This doesn't cause any problems (if you don't mind slow responses) but why maintain a system of expensive machinery to record facts no one is interested in?

Now, with an understanding of the importance of sources, we can make a fairly definitive generalization of what role they should play in the inventory process.

The inventorying of sources and their transaction profiles is a parallel process to the database inventory. The database inventory describes what facts we are trying to maintain - the source inventory describes what facts we are getting and using. The process of building the source inventory is thus one of attempting to match the activity we know must occur with the people and things that generate it. Using the QDD facilities, this can be done in two ways.

The first, if we have enough information at the outset, would be to enumerate all known sources and their transmission rates and assign to each with a probability of 1 the TTD STM. Then, as we learn more about each sources activity, we can begin to definitize

the TTD's while, at the same time, appropriately reducing the amount of activity for which the stochastic model is responsible. For example, if our stochastic model shows

R(C, PLANES) = 10 minutes

P(C, UTM-GRID-NO | C, PLANES) = .8

and we discover a tracking station for which the TTD inter-event time is 36 minutes and the TTD is defined to be

C PLANES/UTM-GRID-NO WHERE ID = 1

then by adding this to the source profile, we would also be justified in reducing R(C, PLANES) to 15 minutes since 1/3 of the originally observed activity has now been accounted for by a source. By persisting in this analysis, we hope to reach a point at which either

(1) the stochastic activity is zero

(2) we can confirm that there is necessary activity that is simply not being generated.

The second approach does not assume any initial information about the sources is available. Instead, we use the stochastic model as a guideline for investigating sources to determine how, and in what amount, they relate.

We begin by hypothesizing that a source of activity must exist for each event described in the stochastic model. This hypothesis

is stated, in QDD, by converting every independent event to a corresponding TTD, and assigning each result TTD to a unique source. For example, if the stochastic model defined

R(A,PLANES) = 12 hours,

we would replace this by the TTD

A S1 OF PLANES WHERE QLF

and create a hypothetical source, which we might conveniently name A-PLANES, and assert that the mean inter-event time for this TTD being transmitted by the source is 12 hours.

The next step is to use this hypothesized source inventory as a baseline profile, and to define each real source with respect to it. This process is carried out by either collocating or decomposing the TTD's according to what we learn.

Collocation is the process of lumping activity together when we identify a source that is performing more than one independent event. For example, assume we had two hypothesized sources:

A-PLANES transmitting TTD1 of A S1 OF PLANES WHERE QLF;

A-CREW transmitting TTD2 of A S2 OF CREW WHERE QLF.

but suppose our investigation reveals a real source, say BASECOM which, in fact, assigns both planes and crew and, moreover, does so concurrently whenever a new plane is assigned. We would then

collocate TTD1 and TTD2 under the new source, BASECOM, by asserting that

BASECOM transmits TTD1 independently

If BASECOM transmits TTD1, it will transmit TTD2 with probability of 1.

and deleting the hypothesized A-PLANES and A-CREW sources.

Decomposition is the process of breaking activity apart when we identify that more than one real sources is performing it. This can occur quantitatively, qualitatively, or both. Quantitative decomposition is self-explanatory; we identify two (or more) sources doing the same thing functionally, but each at a lesser rate than dictated by the stochastic model. For example, we find two sources, BASECOM and HEADQUARTERS that both assign planes independently and with equal probability. We would consequently replace

A-PLANES transmits TTD1 every 12 hours

with

BASECOM transmits TTD1 every 24 hours;

HEADQUARTERS transmits TTD1 every 24 hours.

A qualitative decomposition is necessary when we find not only that two sources are sharing the responsibility for an event, but also that the division has a functional delineation. By the previous example, suppose we found that HEADQUARTERS assigned tactical

aircraft (fighters and bombers) and that BASECOM assigned logistical aircraft (transports, recon and rescue). We would represent this by the qualitative decomposition of

TTD1: A S1 OF PLANES WHERE QLF

into

TTD1A: A S1 OF PLANES WHERE TYPE = 2 AND QLF

TTD1B: A S1 OF PLANES WHERE TYPE = 3 AND QLF

and the appropriate assignment of each to its respective source.

In this approach to source inventory, we continue the process of redistributing functional activity among sources until either

(1) all of it has been allocated, or

(2) we can confirm that some of the functional activity is not taking place.

## 28.0 USING QDD FORMS

While QDD can serve as a philosophy with which to approach a database inventory effort, it can also serve directly as an analytic tool. This is accomplished by using the seven QDD FORMS provided in Appendix A as a database inventory record. These forms not only provide a convenient means of organizing the inventory data, they also act as a checklist to discipline the inventory process. Following is a description of each form and how to encode the pertinent information.

### The Relation Form

The Relation Form is used to record information directly pertinent to the relations. And since the relations represent the principal entities tracked by the database, the Relation Form is generally the first to be completed.

Each relation identified is entered by establishing a unique RELATION NAME. In addition, each relation must be given a Name-In-Context Name, or NICNAME, for use as a reference on the other forms. The number of tuples in the relation, #(r), is entered in exponential notation both to accommodate widely differing magnitudes and to afford quick cursory analysis. The independent rates R(e,r), are entered in the appropriate column according to e and exponential

inter-event time in seconds is the numeric notation. The remainder of the form is available for remarks and a sequence number in the event it is desired to automate the inventory data.

The Attribute Form

When an attribute of a relation is identified, it is established by entering an ATTRIBUTE NAME on the Attribute Form along with the RELATION NICNAME to which it belongs. The ATTRIBUTE NAME need only be unique within the relation. The attribute, too, must be given a NICNAME for reference. The population of the attribute is entered in exponential notation and denotes the number of unique values the attribute takes on in the relation.

The RELATION DISTRIBUTION is a vector that describes the way in which the tuples of the relation are distributed about the values in the attribute. The entry is prepared by ordering the attribute values according to the frequency with which they're observed in tuples (highest frequency first) and then:

(1) dividing the attribute values into five divisions of 20% each ;

(2) computing the average frequency for each division;

(3) converting this average frequency to a percentage of the tuples;

(4) entering this percentage in the column corresponding to the division.

The DOMAIN NICNAME indicates from which underlying domain the attribute obtains its value. The corresponding DOMAIN DISTRIBUTION is a vector that describes the way in which the attribute values are distributed in the domain. The entry is prepared by ordering the domain values by the frequency with which they're observed in the relation (highest frequency first) and then performing steps (1) - (4) above with step (1) modified to (1) dividing the domain values into five divisions of 20% each.

The Domain Form

Each identified underlying domain is established by entering a unique DOMAIN NAME and a NICNAME for reference. The population is determined by enumerating the superset of all attribute values, without regard for relation, that draw their values from the domain.

The Event Form

The Event Form is the vehicle for recording declarations of the form:

$$T(e,a,r \mid e_1,a_1,r_1) = V.$$

This generalization allows at least all First-Order dependencies and Second-Order biases. But as with most generalizations, it also permits combinations that don't make sense (yet). For this reason, there are some rules that must be followed.

The Dependent Event corressponds to the left half of the declaration. The RELATION NICNAME is required but the ATTRIBUTE NICNAME may or may not be present - its absence denotes that the event is with respect to the entire relation.

The Independent Event corresponds to the right half of the expression and the same rule applies to ATTRIBUTE NICNAME.

The TYPE denotes whether the event is a First Order Dependency (denoted 1) or a Second Order Bias (denoted 2).

The Value is the decimal fraction that represents the probability or bias and if TYPE = 2, the value must be signed (+ or -).

The VECTOR is used in lieu of VALUE only for biases (TYPE = 2) to the distribution and the V1 - V5 are signed decimal fractions.

The permissible combinations are as follows:

| DEPENDENT EVENT | | INDEPENDENT EVENT | | TYPE | VALUE | VECTOR | OTHER RULES |
|---|---|---|---|---|---|---|---|
| EVENT TYPE | ATTRIBUTE PRESENT | EVENT TYPE | ATTRIBUTE PRESENT | | | | |
| A,D,C,R | YES | A,D,C,R | YES | 2 | YES | NO | RELATION & EVENT TYPE MUST BE THE SAME |
| A,D,C,R | YES | A,D,C,R | NO | 1 | YES | NO | RELATION & EVENT TYPE MUST BE THE SAME |
| A,D | YES | C | NO | 1 | YES | NO | RELATIONS MUST BE THE SAME |
| A,D,C,R | NO | A,D,C,R | NO | 1 or 2 | YES | NO | RELATIONS MUST HAVE A COMMON DOMAIN |
| Q | YES | Q | YES | 2 | YES | NO | RELATIONS MUST BE THE SAME |
| Q | YES | A,D,C,R | NO | 1 | YES | NO | RELATIONS MUST BE THE SAME |
| Y | YES | Q | YES | 2 | NO | YES | RELATIONS MUST BE THE SAME |
| X | YES | Q | YES | 2 | NO | YES | RELATIONS MUST BE THE SAME |
| Q | NO | Q | YES | 1 | YES | NO | DEPENDENT RELATIONS MUST BE CONSTANT |
| | | | | | | | |

The Source Form

Other than remarks, the only purpose of the Source Form is an exitence list. The existence of a Source is established by entering a SOURCE NAME and NICNAME for reference by other forms.

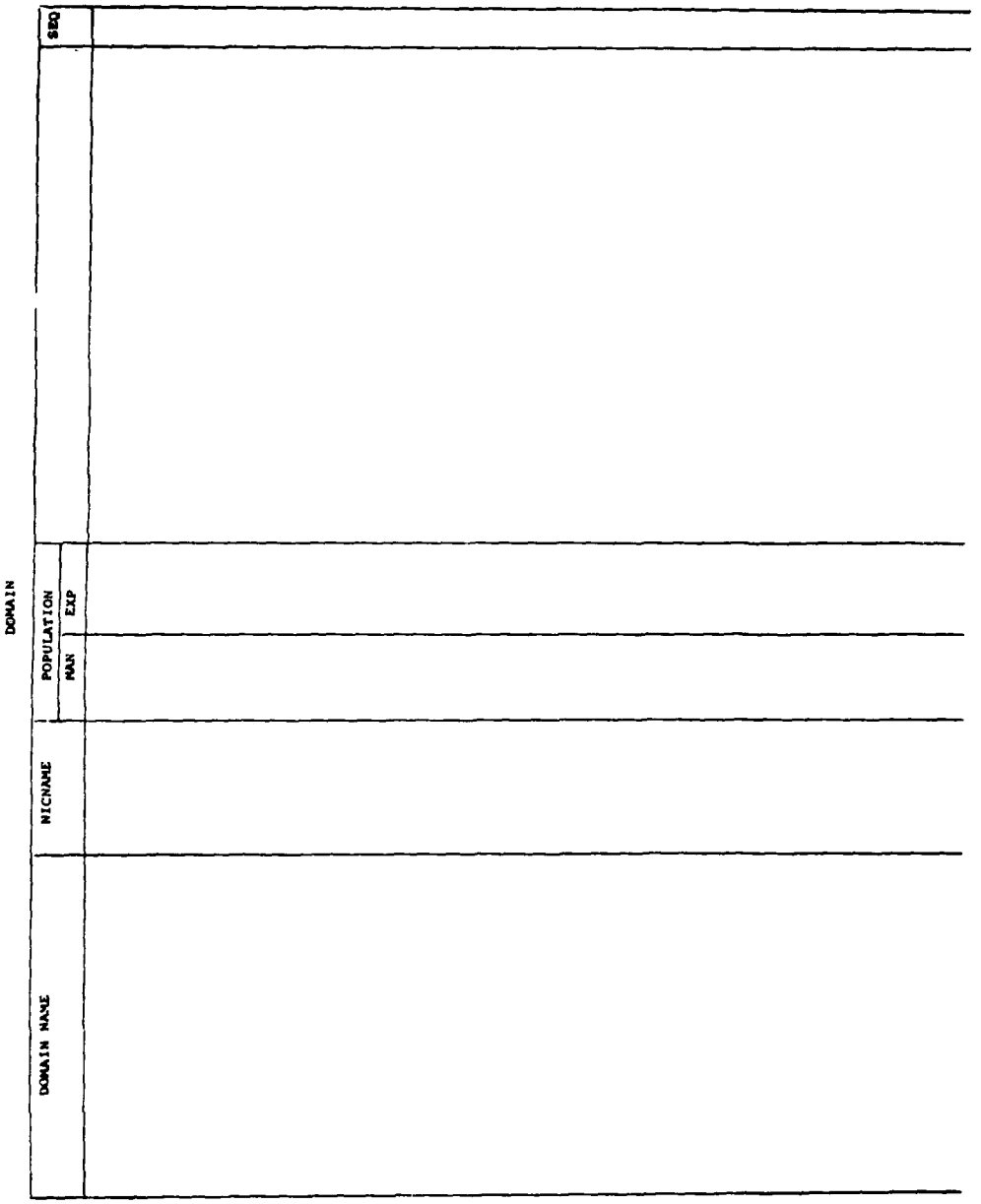
The Transaction Type Form

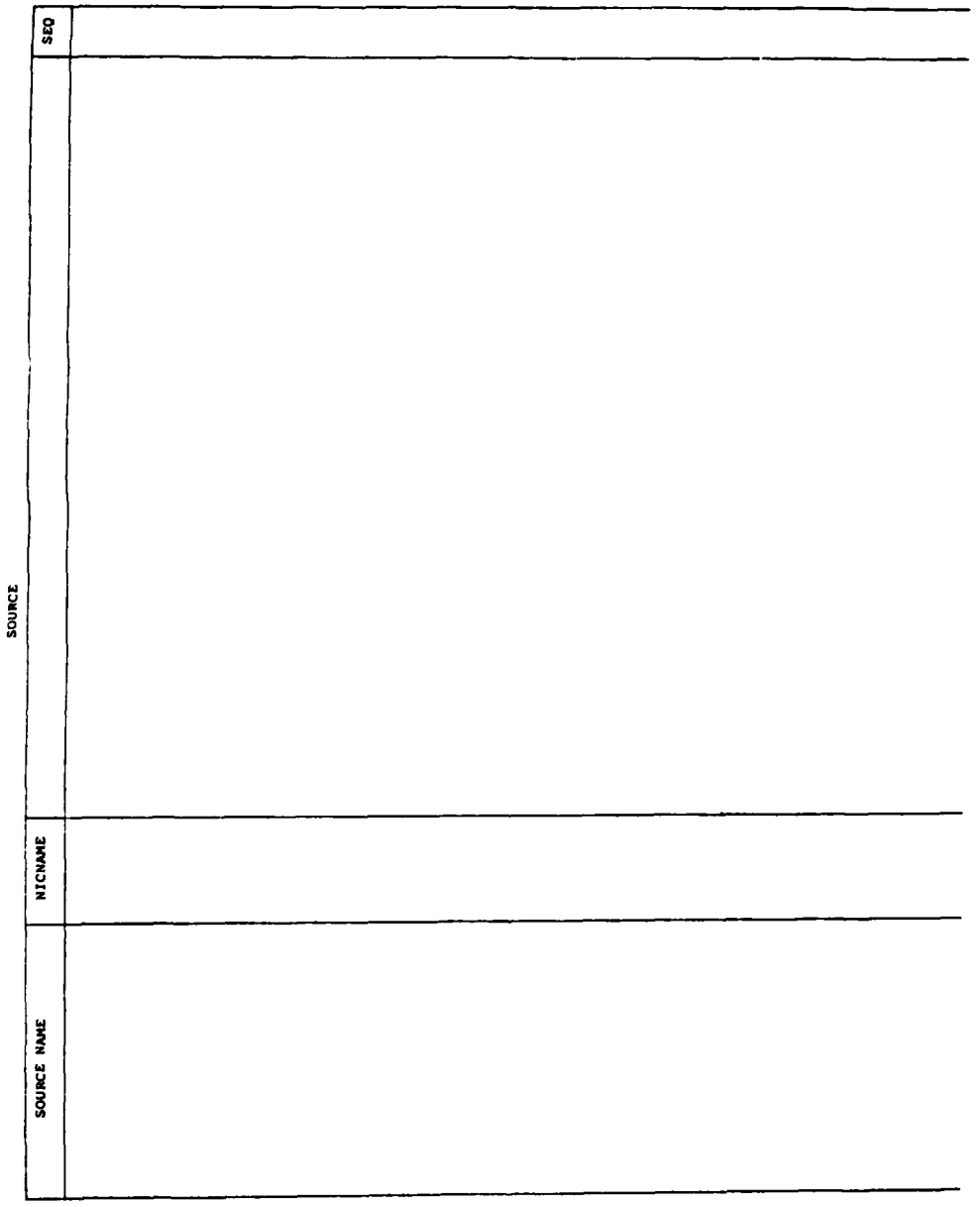
The Transaction Type Form is used to enter RSD statements. When a statement is entered, it is given a TTD NAME and NICNAME for reference.

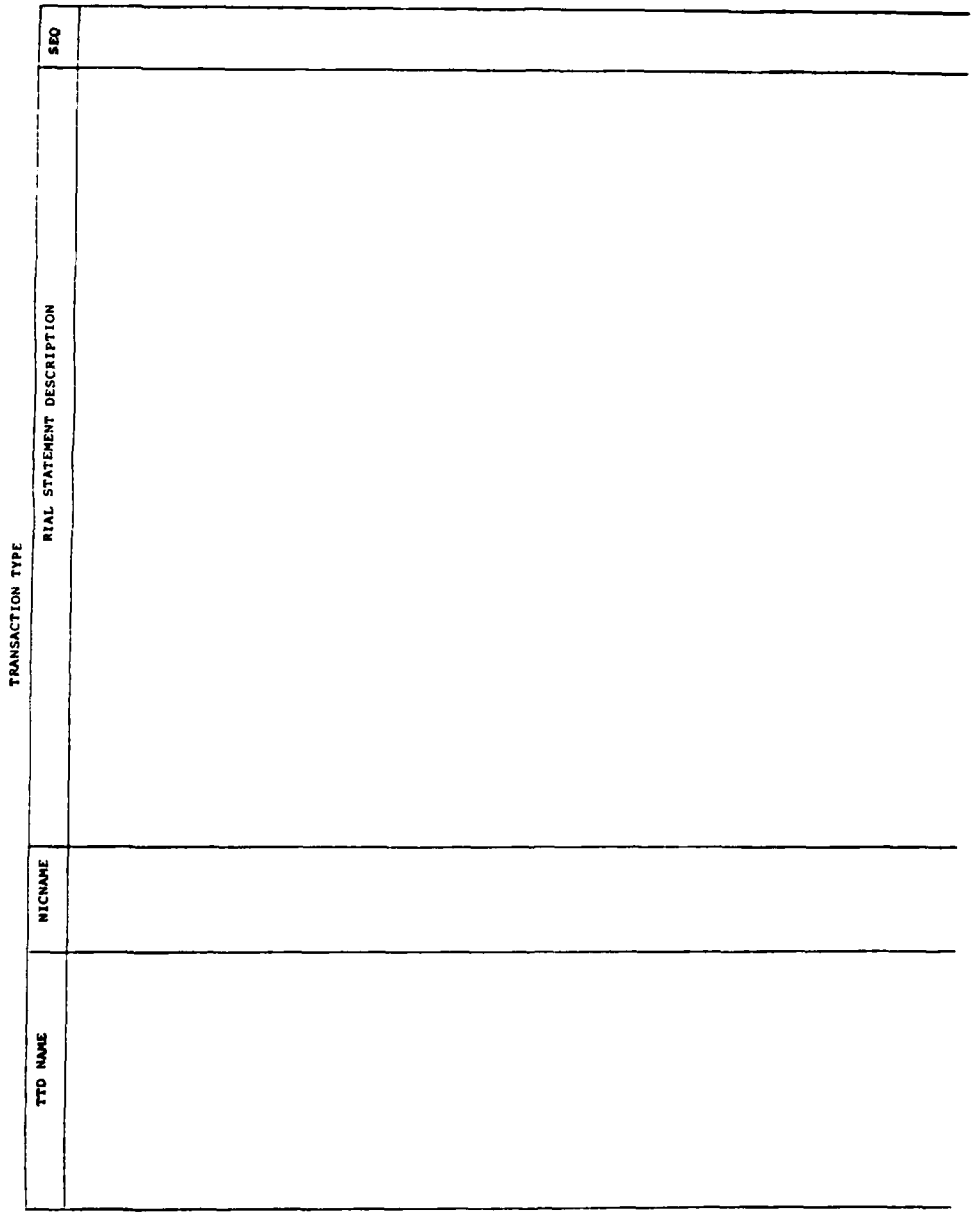
The Activity Form

The Activity Form established the correlation between Sources and the Transactions they transmit. The SOURCE NICNAME identifies the source of the activity. The TTD NICNAME identifies the message transmitted by the Source. The IND RATE in mean inter-event time exponential notation represents the rate at which the Source transmits the message independently of other messages. The DEP PROB represents the probability that the Source will transmit the TTD given that it has transmitted the PREVIOUS TTD NICNAME.

# APPENDIX A


DOMAIN
DOMAIN NAME
NICNAME
POPULATION
MAN
EXP
SBQ


SEQ
SOURCE
NICNAME
SOURCE NAME


TRANSACTION TYPE
RIAL STATEMENT DESCRIPTION
SEQ
NICNAME
TTD NAME

ACTIVITY

| SOURCE NICNAME | TTD NICNAME | IND RATE SECONDS | | PREVIOUS TTD NICNAME | DEP PROB | SEQ |
|---|---|---|---|---|---|---|
| | | MAN | EXP | | | |
| | | | | | | |

EVENT

| DEPENDENT EVENT | | | INDEPENDENT EVENT | | | TYPE | VALUE | VECTOR | | | | | | SEQ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| EVENT TYPE | RELATION NICNAME | ATTRIBUTE NICNAME | EVENT TYPE | RELATION NICNAME | ATTRIBUTE NICNAME | | | V1 | V2 | V3 | V4 | V5 | | |
| | | | | | | | | | | | | | | |

ATTRIBUTE

| ATTRIBUTE NAME | NICKNAME | POPULATION | | RELATION NICKNAME | RELATION DISTRIBUTION (PERCENT BY PERCENTILE) | | | | | DOMAIN NICKNAME | DOMAIN DISTRIBUTION (PERCENT BY PERCENTILE) | | | | | SEQ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | MAN | EXP | | 80th | 60th | 40th | 20th | ZERO | | 80th | 60th | 40th | 20th | ZERO | |
| | | | | | | | | | | | | | | | | |

RELATION

| RELATION NAME | NICNAME | POPULATION | | INDEPENDENT RATE (SECONDS) | | | | | | | | | SEQ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | MAN | EXP | ARRIVE | | DEPART | | CHANGE | | RETRIEVE | | | |
| | | | | MAN | EXP | MAN | EXP | MAN | EXP | MAN | EXP | | |
| | | | | | | | | | | | | | |